

# **EControl Form Designer Pro**

# Table of Contents

<b>EControl Form Designer Pro</b>	<b>1</b>
<b>Overview</b>	<b>1</b>
<b>Features</b>	<b>2</b>
<b>Insallation</b>	<b>3</b>
<b>Using DesignIDE.BPL</b>	<b>5</b>
<b>Integration with scripters</b>	<b>6</b>
<b>License</b>	<b>8</b>
<b>Change Log</b>	<b>12</b>
Version 2.00	12
Version 2.10	14
Version 2.20	15
Version 2.30	17
Version 2.40	17
Version 2.50	17
<b>ecBtnPanel Namespace</b>	<b>18</b>
Classes	18
TBtnMargins Class	18
TBtnMargins Properties	19
TCustomBtnPanel Class	20
TCustomBtnPanel Methods	22
TCustomBtnPanel Properties	24
TCustomBtnPanel Events	27
TBtnPanel Class	28
TBtnPanel Properties	32
Structs, Records, Enums	44
ecBtnPanel.TRowOrientation Enumeration	44
Types	45
ecBtnPanel.TButtonClickEvent Type	45
ecBtnPanel.TDrawButtonEvent Type	45
ecBtnPanel.TGetButtonHintEvent Type	45
<b>ecDIList Namespace</b>	<b>45</b>
Classes	46
TCustomPropList Class	46
TCustomPropList Methods	49
TCustomPropList Properties	50

TCustomPropList Events	51
TDualList Class	52
TDualList Methods	54
TDualList Properties	58
TPropertyItem Class	61
TPropertyItem Methods	62
TPropertyItem Properties	64
TPropListRoot Class	65
TPropListRoot Methods	67
TPropListRoot Properties	68
Structs, Records, Enums	69
ecDList.TCellType Enumeration	69
Types	69
ecDList.TCustomPropDrawEvent Type	70
ecDList.TGetCellParamsEvent Type	70
<b>ecExtEdit Namespace</b>	<b>70</b>
Classes	70
TBtnEdit Class	70
TBtnEdit Methods	73
TBtnEdit Properties	76
TBtnEdit Events	78
TCustomEditEx Class	78
TCustomEditEx Methods	82
TCustomEditEx Properties	84
TCustomEditEx Events	85
TEditEx Class	85
TEditEx Properties	92
TPopupListbox Class	105
TPopupListbox Methods	106
TPopupListbox Properties	106
TUnicodeEdit Class	107
TUnicodeEdit Methods	108
TUnicodeEdit Properties	108
Structs, Records, Enums	109
ecExtEdit.TInplaceEditStyle Enumeration	110
Types	110
ecExtEdit.TCloseUpEvent Type	110
ecExtEdit.TMeasureWidthEvent Type	110
ecExtEdit.TOnAcceptListValueEvent Type	111
<b>ecHintHelper Namespace</b>	<b>111</b>
Classes	111

TecHintHelper Class	111
TecHintHelper Methods	112
TecHintHelper Properties	113
Structs, Records, Enums	114
ecHintHelper.TCMGetHintData Record	114
ecHintHelper.TecHintData Record	115
Types	115
ecHintHelper.PecHintData Type	115
Constants	115
ecHintHelper.CM_GETHINTDATA Constant	116
<b>ecToolList Namespace</b>	<b>116</b>
Classes	116
TCustomToolList Class	116
TCustomToolList Methods	118
TCustomToolList Properties	120
TCustomToolList Events	123
TToolItemStyle Class	123
TToolItemStyle Methods	124
TToolItemStyle Properties	124
TToolItemStyle Events	125
TToolList Class	125
TToolList Properties	130
TToolListItem Class	141
TToolListItem Properties	142
TToolListItems Class	143
TToolListItems Properties	143
Structs, Records, Enums	143
ecToolList.TItemShape Enumeration	144
Types	144
ecToolList.TToolItemState Type	144
<b>ed_DsnBase Namespace</b>	<b>144</b>
Classes	144
TBaseDesigner Class	144
TBaseDesigner Methods	146
TBaseDesigner Properties	151
TBaseDesigner Events	152
Structs, Records, Enums	155
ed_DsnBase.TDesignOperation Enumeration	155
ed_DsnBase.TDsnDragState Enumeration	155
Types	156
ed_DsnBase.TDesignOperations Type	156

ed_DsnBase.THandleControlMessage Type	156
<b>ed_Designer Namespace</b>	<b>156</b>
Classes	156
TControlGroups Class	156
TControlGroups Methods	157
TControlGroups Properties	158
TPasteInfo Class	159
TPasteInfo Methods	159
TPasteInfo Properties	160
TzCustomFormDesigner Class	161
TzCustomFormDesigner Methods	170
TzCustomFormDesigner Properties	187
TzCustomFormDesigner Events	194
TzFormDesigner Class	201
TzFormDesigner Properties	214
Structs, Records, Enums	229
ed_Designer.TBufferizedType Enumeration	229
ed_Designer.TCompAlign Enumeration	229
ed_Designer.TCompSize Enumeration	230
ed_Designer.TGuidelinesStyle Enumeration	230
ed_Designer.TLocalMenuFilter Enumeration	231
Types	231
ed_Designer.TComponentEvent Type	232
ed_Designer.TCreateComponentEvent Type	232
ed_Designer.TCreateFrameEvent Type	232
ed_Designer.TCreateIconEvent Type	232
ed_Designer.TCreateMethodEvent Type	233
ed_Designer.TDrawControlEvent Type	233
ed_Designer.TGetComponentHintEvent Type	233
ed_Designer.TGetMethodNamesEvent Type	233
ed_Designer.TGetObjNameEvent Type	233
ed_Designer.TGetScriptProcEvent Type	234
ed_Designer.TGuidelinesStyles Type	234
ed_Designer.THandleActionEvent Type	234
ed_Designer.TLocalMenuFilters Type	234
ed_Designer.TNotificationEvent Type	234
ed_Designer.TRenameEvent Type	235
ed_Designer.TRenameMethodEvent Type	235
ed_Designer.TSetNameEvent Type	235
ed_Designer.TSetScriptProcEvent Type	235
ed_Designer.TShowMethodEvent Type	235
ed_Designer.TUndoRecEvent Type	236

ed_Designer.TValidateMethodEvent Type	236
Constants	236
ed_Designer.DM_POSCHANGED Constant	236
ed_Designer.sLineBreak Constant	236
<b>ed_dsncont Namespace</b>	<b>237</b>
Classes	237
TDesignSurface Class	237
TDesignSurface Methods	238
TDesignSurface Properties	239
Structs, Records, Enums	241
ed_dsncont.TRulerUnits Enumeration	241
<b>ed_RegComps Namespace</b>	<b>242</b>
Classes	242
Frames Class	242
TComponentClassInfo Class	242
TComponentClassInfo Methods	243
TComponentClassInfo Properties	244
TCustomModuleInfo Class	245
TFrameInfo Class	245
TFrameInfo Properties	246
TPackageInfo Class	246
TPackageInfo Methods	247
TPackageInfo Properties	247
TPackageMng Class	248
TPackageMng Methods	250
TPackageMng Properties	255
TPackageMng Events	256
Functions	256
ed_RegComps.DrawBtnIcon Function	257
Structs, Records, Enums	257
ed_RegComps.TIconBtnStyle Enumeration	257
Types	258
ed_RegComps.TComponentRegEvent Type	258
ed_RegComps.TComponentRegInfoEvent Type	258
Variables	258
ed_RegComps.PackageMng Variable	258
ed_RegComps.Runtime Variable	259
<b>ed_RegMeth Namespace</b>	<b>259</b>
Classes	259
TDefaultMethodRegister Class	259
TDefaultMethodRegister Fields	261

TDefaultMethodRegister Methods	261
TDefaultMethodRegister Properties	265
Structs, Records, Enums	265
ed_RegMeth.TMethodInfo Record	265
Types	266
ed_RegMeth.PMethod Type	266
ed_RegMeth.PMethodInfo Type	266
Variables	266
ed_RegMeth.MethRegister Variable	267
<b>ed_ObjTree Namespace</b>	<b>267</b>
Classes	267
TCustomDesignerObjTree Class	267
TCustomDesignerObjTree Methods	269
TCustomDesignerObjTree Properties	271
TCustomDesignerObjTree Events	271
TDesignerObjTree Class	271
TDesignerObjTree Methods	277
TDesignerObjTree Properties	277
Functions	294
ed_ObjTree.CreateGhostedImages Function	294
Types	294
ed_ObjTree.TCreateSprigNodeEvent Type	295
<b>ed_TextEdit Namespace</b>	<b>295</b>
Classes	295
TDsnInplaceEditor Class	295
TDsnInplaceEditor Methods	299
TDsnInplaceEditor Properties	300
TInplaceComponentEditor Class	302
TInplaceComponentEditor Methods	303
TInplaceComponentEditor Properties	305
Functions	306
ed_TextEdit.CreateImplEditor Function	306
ed_TextEdit.RegisterInplaceComponentEditor Function	306
Types	307
ed_TextEdit.TInplaceComponentEditorClass Type	307
<b>edActns Namespace</b>	<b>307</b>
Classes	307
TDesignerAction Class	308
TDesignerAction Methods	309
TDesignerAction Properties	309
TdsnAlignmentDlg Class	311

TdsnAlignmentDlg Methods	313
TdsnAlignToGrid Class	313
TdsnAlignToGrid Methods	314
TdsnBringToFront Class	315
TdsnBringToFront Methods	316
TdsnCopy Class	316
TdsnCreationOrderDlg Class	317
TdsnCreationOrderDlg Methods	318
TdsnCut Class	318
TdsnDelete Class	318
TdsnDesignMode Class	319
TdsnDesignMode Methods	320
TdsnFlipChildren Class	320
TdsnFlipChildren Methods	322
TdsnFlipChildrenAll Class	322
TdsnFlipChildrenAll Methods	323
TdsnGroupControls Class	323
TdsnGroupControls Methods	325
TdsnLockControls Class	325
TdsnLockControls Methods	326
TdsnPaste Class	327
TdsnRedo Class	327
TdsnScale Class	327
TdsnScale Methods	328
TDsnSelAction Class	329
TDsnSelAction Methods	330
TdsnSelectAll Class	330
TdsnSendToBack Class	330
TdsnSendToBack Methods	332
TdsnShowTabOrder Class	332
TdsnShowTabOrder Methods	333
TdsnSizeDlg Class	334
TdsnSizeDlg Methods	335
TdsnTabOrderDlg Class	335
TdsnTabOrderDlg Methods	336
TdsnTargetAction Class	337
TdsnTextEditMode Class	337
TdsnTextEditMode Methods	338
TdsnUndo Class	338
TdsnUngroupControls Class	339
TdsnUngroupControls Methods	340
<b>edcCmbCombo Namespace</b>	<b>340</b>



Classes	340
TComponentCombo Class	341
TComponentCombo Methods	344
TComponentCombo Properties	346
TComponentCombo Events	358
Types	358
edcCmbCombo.TCanAddObjectEvent Type	359
edcCmbCombo.TGetClassNameEvent Type	359
edcCmbCombo.TGetComponentsEvent Type	359
edcCmbCombo.TSelChangedEvent Type	359
<b>edcCompPal Namespace</b>	<b>359</b>
Classes	360
TPalettePanel Class	360
TPalettePanel Methods	364
TPalettePanel Properties	366
TPaletteTab Class	377
TPaletteTab Properties	380
<b>edcDsnEvents Namespace</b>	<b>389</b>
Classes	389
TDesignerEvents Class	389
TDesignerEvents Events	391
Types	394
edcDsnEvents.TDesignerEvent Type	394
edcDsnEvents.TDsnItemEvent Type	395
edcDsnEvents.TDsnKeyDownEvent Type	395
edcDsnEvents.TDsnKeyPressEvent Type	395
edcDsnEvents.TGetGlobalsEvent Type	395
edcDsnEvents.TOnGetPoint Type	395
edcDsnEvents.TRegisterComponentEvent Type	396
<b>edcPropCtrl Namespace</b>	<b>396</b>
Classes	396
TCategoryNode Class	396
TCategoryNode Methods	398
TCustomInspectorList Class	399
TCustomInspectorList Methods	404
TCustomInspectorList Properties	407
TCustomInspectorList Events	412
TInspectorList Class	413
TInspectorList Properties	422
TPropertyNode Class	439
TPropertyNode Methods	441

TPropertyNode Properties	441
TPropertyNodes Class	442
TPropertyNodes Methods	444
TzDesignerSelections Class	445
Structs, Records, Enums	445
edcPropCtrl.TTypeSelector Enumeration	446
Types	446
edcPropCtrl.IFormDesigner Type	446
edcPropCtrl.IProperty Type	446
edcPropCtrl.TAcceptCategoryEvent Type	447
edcPropCtrl.TAcceptPropertyEvent Type	447
edcPropCtrl.TChangeSelectionEvent Type	447
edcPropCtrl.TGetPropReadOnlyEvent Type	447
edcPropCtrl.TOnInspSetPropValueEventA Type	447
edcPropCtrl.TOnInspSetPropValueEventW Type	448
<b>edcPropEdit Namespace</b>	<b>448</b>
Classes	448
TCustomPropertyEdit Class	448
TCustomPropertyEdit Methods	453
TCustomPropertyEdit Properties	455
TCustomPropertyEdit Events	456
TPropertyEdit Class	457
TPropertyEdit Properties	462
TPropertyNameProperty Class	463
Interfaces	464
IPropertyStatusImage Interface	464
IPropertyStatusImage Methods	464
Types	465
edcPropEdit.TOnSetPropValueEventA Type	465
edcPropEdit.TOnSetPropValueEventW Type	465
<b>eddAlignDlg Namespace</b>	<b>465</b>
Classes	465
TAlignmentDlg Class	465
<b>eddAlignPal Namespace</b>	<b>466</b>
Classes	466
TAlignPalette Class	466
<b>eddCrOrdDI Namespace</b>	<b>467</b>
Classes	467
TCreateOrderDlg Class	468
<b>eddCustomPal Namespace</b>	<b>468</b>

Classes	468
TCustomizePaletteDlg Class	468
<b>edcToolList Namespace</b>	<b>469</b>
Classes	470
TPaletteToolList Class	470
TPaletteToolList Methods	476
TPaletteToolList Properties	477
TPaletteToolList Events	489
<b>eddDsnOpt Namespace</b>	<b>489</b>
Classes	490
TDsnOptionsDlg Class	490
TDsnOptionsDlg Properties	491
<b>eddObjInspProp Namespace</b>	<b>491</b>
Classes	491
TObjInspPropDlg Class	491
TObjInspPropDlg Properties	493
<b>eddPackageCtrl Namespace</b>	<b>493</b>
Classes	493
TPackageCtrlDlg Class	493
<b>eddPageName Namespace</b>	<b>494</b>
Classes	494
TPageNameDlg Class	494
<b>eddObjInspFrm Namespace</b>	<b>494</b>
Classes	494
TObjectInspectorFrame Class	495
TObjectInspectorFrame Methods	496
TObjectInspectorFrame Properties	497
TObjectInspectorFrame Events	497
Types	498
eddObjInspFrm.TObjInspTabs Type	498
<b>eddScaleDI Namespace</b>	<b>498</b>
Classes	498
TScaleDlg Class	498
<b>eddSelFrame Namespace</b>	<b>499</b>
Classes	499
TSelFrameDlg Class	499
<b>eddObjTreeFrame Namespace</b>	<b>499</b>
Classes	500
TObjectTreeFrame Class	500

TObjectTreeFrame Methods	500
TObjectTreeFrame Properties	501
<b>eddSizeDlg Namespace</b>	<b>501</b>
Classes	501
TSizeAdjDlg Class	501
<b>eddTabOrdDI Namespace</b>	<b>502</b>
Classes	502
TTabOrderDlg Class	502
<b>edlOUtils Namespace</b>	<b>502</b>
Functions	503
edlOUtils.zCopyCmpResource Function	504
edlOUtils.zReadCmpFromFile Function	504
edlOUtils.zReadCmpFromStream Function	504
edlOUtils.zWriteCmpToFile Function	505
edlOUtils.zWriteCmpToStream Function	505
<b>edManager Namespace</b>	<b>505</b>
Classes	505
TDesignerManager Class	505
TDesignerManager Methods	507
TDesignerManager Properties	510
Interfaces	511
IClassSelector Interface	511
IClassSelector Methods	511
IDesignIDEEEvents Interface	512
IDesignIDEEEvents Methods	512
Functions	513
edManager.GetClassDragImage Function	513
Structs, Records, Enums	513
edManager.TComponentClassDragImage Enumeration	513
Variables	514
edManager.DsnManager Variable	514
<b>edsMenuDsn Namespace</b>	<b>514</b>
Classes	514
TMenuDsnWnd Class	514
TzMenuEditor Class	515
TzMenuItemsPropertyEditor Class	515
<b>eduDMContainer Namespace</b>	<b>515</b>
Classes	515
TDsnDM Class	516

<b>eduServObj Namespace</b>	<b>516</b>
Classes	516
TAlignRuler Class	516
TComponentCaption Class	517
TComponentIcon Class	517
TDraggedControl Class	517
TSmallRect Class	517
TTabOrderIcons Class	518
TTabOrderIcons Methods	519
TTabOrderIcons Properties	519
TzBoundCtrl Class	520
TzBoundCtrl Methods	521
TzBoundCtrl Properties	521
Functions	523
eduServObj.DrawPatternRect Function	523
eduServObj.IsServiceControl Function	523
Structs, Records, Enums	523
eduServObj.TMarkerShape Enumeration	524
eduServObj.TVerticalAlignment Enumeration	524
<b>edUtils Namespace</b>	<b>524</b>
Functions	524
edUtils.DsnAlignSelected Function	525
edUtils.DsnLoadPackage Function	525
edUtils.DsnReadCmpFromStream Function	525
edUtils.DsnReadFromFile Function	525
edUtils.DsnWriteCmpToStream Function	526
edUtils.DsnWriteToFile Function	526
edUtils.GetDesigner Function	526
edUtils.InvalidateControl Function	526
edUtils.IsControlParent Function	526
edUtils.NormalizeRect Function	527
edUtils.PerformDsnAction Function	527
edUtils.ShowDesignerOptionsDlg Function	527
edUtils.ShowDsnAbout Function	527
Structs, Records, Enums	527
edUtils.TDesignerAction Enumeration	528

## Index

## a

# 1 EControl Form Designer Pro

## 1.1 Overview

**EControl Form Designer Pro** - powerful form designer which is based on the same concepts as Delphi's form designer. It has similar to Delphi interface.

Developed for:

**Delphi:** 5, 6, 7, 2005, 2006, 2007, 2009, 2010, 2011 (XE), 2012 (XE2)

**C++Builder:** 5, 6, 2006, 2007, 2009, 2010, 2011 (XE), 2012 (XE2)

EControl Form Designer Pro has **all required tools**:

Designer (TzFormDesigner),

Design surface (TDesignSurface),

Object inspector (TInspectorList, TObjectInspector),

Component palette (TPalettePanel, TPaletteTab),

Component combo box (TComponentCombo),

Tools palette (TPaletteToolList),

Object tree view (TDesignerObjTree),

Design time dialogs (Align, Align Palette, Size, Scale, Tab order, Creation order, etc.),

Component palette and packages managing dialogs.

Using EControl Form Designer Pro you may easily create and integrate design environment in your applications. It uses all registered property editors, component editors and other design objects. You may use all abilities of existed design objects which are essential part of any VCL library.

Software branches where EControl Form Designer Pro will be very useful:

1. SCADA applications, Industrial automation.
2. Database applications which need to be extendable by users.
3. Engineering applications.
4. All others branches where runtime extensions are required.

**Simple integration with scripts** allows creating full featured IDE.

EControl Form Designer Pro may be easily integrated with different Scripters.

There is a set of events that allow to handle events assignment. In "Integration with scripters (see page 6)" you may read

detailed description of this task by the example of integration with Fast Script library.

EControl Form Designer Pro allows editing of **different targets**:

Forms (any object derived from TForm),

Frames (any object derived from TFrame),

Data Modules (any object derived from TDataModule),

Control based classes (for example, quick report),

Only part of the form (for example, panel or tab sheet).

EControl Form Designer Pro is **easy to use** - just only drop TzFormDesigner, select Target and activate.

---

## 1.2 Features

### Main features

- Fully compatible with Delphi IDE
- It is based on the same interfaces like Delphi form designer
- You can edit forms, data modules, reports, controls on you form
- Full Unicode support.
- Undo, Redo operations.
- Design Surface - container for designed forms.
- Im-place text editing - original feature which provides ability of editing text captions directly on control. This makes designing process more comfortable.
- Object inspector looks and works like Delphi's one and now has BDS style with support of property categories, gutter, references, Unicode properties edit...
- State images - special interface to add images to property edit control at the left side.
- Easy integration with 3D-party Scripters

### Other features

- Support custom modules
- Support property and component editors
- Collection, picture, string, dataset fields, actions and other editors are available
- "Object Inspector Properties" dialog.
- Cross modules references (object and method)
- Event properties editing
- Runtime package loading
- Palette panel for selecting Class to insert
- Palette tab control like Delphi one
- Alignment palette
- Align, Size, Scale, Tab Order, Creation Order dialogs

- Designer hints
- Clipboard operations compatible with Delphi (you can copy components to/ from Delphi)
- Customizable performance
- "Packages" dialog allows Load/Unload packages.
- "Customize Palette" dialog allows customizing component palette.
- Save/Load component palette and packages configuration.
- Frame support. You can insert and edit frames.
- Properties Root, Form, ContainerWindow are now public. To set design target Published property Target has been added.
- Property Edit - edit control for editing property value using property editors.
- BDS style design operations
- Object Tree View - displays a tree diagram of the visual and nonvisual components you place on a form, data module, or frame.
- Tools palette list - Tool list with component palette. Allows in-place rearranging, autocollapsing, vertical categories, components filtration, etc.
- Ready-to-use frames: object inspector frame, component palette tool list frame, object tree view frame - encapsulate default behavior and make creation of design environment faster.
- ...

## 1.3 Installation

EControl Form Designer Pro consist of only one package: zDesignXX.bpl

### 1. Unpack source files, for example, in "C:\EControl".

There will be 6 directories:

C:\EControl\EControl\_Designer\Packages

C:\EControl\EControl\_Designer\Sources

C:\EControl\EControl\_Designer\No\_Bpl

C:\EControl\EControl\_Designer\DsnEditors

C:\EControl\EControl\_Designer\Demo

C:\EControl\EControl\_Designer\Help

### 2. Update library path:

- Open in Delphi IDE dialog "Environment Options", tab sheet "Library"

- Add to "Library path"

- C:\EControl\EControl\_Designer\Sources;
- C:\EControl\EControl\_Designer\No\_Bpl;
- C:\EControl\EControl\_Designer\No\_Bpl\DB;
- C:\EControl\EControl\_Designer\No\_Bpl\DB\Extra;
- C:\EControl\EControl\_Designer\DsnEditors;
- \$(DELPHI)\Source\ToolsApi in Delphi 5,6,7 or \$(DELPHI)\Source\Win32\ToolsApi in BDS;



- **\$(DELPHI)\Source\Property editors** in Delphi 5,6,7 or **\$(DELPHI)\Source\Win32\Property editors** in BDS.

### 3. Open library project (EControl\_Designer\Packages\):

zDesign5.dpk - Delphi 5;  
 zDesign6.dpk - Delphi 6;  
 zDesign7.dpk - Delphi 7;  
 zDesign9.bdsproj - Delphi 2005;  
 zDesign10.bdsproj - Delphi 2006;  
 zDesign11.dproj - Delphi 2007;  
 zDesign12.dproj - Delphi 2009;  
 zDesign14.dproj - Delphi 2010;  
 zDesign15.dproj - Delphi XE;  
 zDesign16.dproj - Delphi XE2, C++Builder XE2;  
 zDesignC5.bpk - C++Builder 5;  
 zDesignC6.bpk - C++Builder 6;  
 zDesign10C.bdsproj - C++Builder 2006;  
 zDesign11C.cbproj - C++Builder 2007  
 zDesign12C.cbproj - C++Builder 2009  
 zDesign14C.cbproj - C++Builder 2010  
 zDesign15C.cbproj - C++Builder XE

### 4. Install package.

In project popup menu select command "Install". Library will be compiled and installed.

### 5. Important conditionals (you should add them in project options)

**EC\_NO\_BPL** - added for applications without using runtime packages,

**EC\_DSN\_REG** - removes registration confirmation,

**NEWNEWDESIGNER** - required for applications without using runtime packages to compile TStringListProperty (unit stredit).

### 6. Building applications

EControl Form Designer Pro may be used in applications that are built with runtime packages and in applications without using runtime packages, i.e. in standalone EXE. See Using DesignIDE.BPL (see page 5) topic for more details.

### 7. TRIAL version.

Trial version can be used only with runtime packages! To build any application you need to update project options (see Using DesignIDE.BPL (see page 5) topic for more details).

Archive with trial version contains next files: zDesign##.bpl, zDesign##.dcp for Delphi; zDesignC##.bpl, zDesignC##.bpi, zDesignC##.lib for C++Builder.

Unpack these files to "...BPL" and "...DCP" folders (depends on Delphi/C++Builder version). In IDE select "Component | Install packages" menu where add zDesign##.bpl to installed packages.

### 8. C++ Builder specific

Concept of using Form Designer without runtime packages is based on recompilation of designIDE units from (\$BDS)\Source\ToolsAPI; (\$BDS)\Source\Win32\Property Editors and EControl\_Designer\No\_Bpl with global define EC\_NO\_BPL. These files are Pascal units. In Delphi recompilation is performed automatically, but in C++Builder these units will not be recompiled. Object files created when building package do not suit because they are compiled without EC\_NO\_BPL define.

To resolve this problem you may use next approach: add to your C++ project one pascal file which will use units from folders mentions above, for example:

```
unit Pas_In_Cpp;

interface

{$ObjExportAll On}

uses {$IFDEF VER130}
    DesignWindows,
    ComponentDesigner,
    PropertyCategories,
    edsCompDsn,
    {$ENDIF}
    proxies,
    ed_Designer,
    StdRegComps,
    events,
    ecDllList,
    edActns,
    edcCmbCombo,
    edcCompPal,
    edcPropCtrl,
    picEdit,
    eddObjInspFrm;

{$IFDEF EC_NO_BPL}
ERROR: This unit only to be used when compiling without runtime packages
{$ENDIF}

implementation

end.
```

This approach is used in demo EControl\_Designer\Demo\MDI\_CPP

## 1.4 Using DesignIDE.BPL

EControl Form Designer Pro may be used in applications that are built with runtime packages and in applications without using runtime packages, i.e. in standalone EXE.

### Using with runtime packages.

In this mode you may load design time packages at runtime. All design time classes will be registered automatically.

Advantages - automatic design objects registration, registration of not available in sources design objects, ability to extend at runtime component base by loading design time packages.

There is only one disadvantage - you need to distribute **DesignIDEXX.bpl**. Distribution of this package is limited by Borland (CodeGear) License (see page 8).

Such using is possible if you will use application on the computers where Delphi is installed.

**Using without runtime packages.**

When you build application without runtime packages all required code is included in executable file. In this case there is no using of **DesignIDEXX.BPL** and there are no licensing and distribution restrictions.

Disadvantages:

1. You should register all design time objects programmatically. Usually you need to call Register methods of libraries units, for example:

```
initialization
  bdereg.Register;
  dbreg.Register;
  cxEditReg.Register;
  cxExtEditReg.Register;
  cxGridReg.Register;
  cxGridPopupMenuReg.Register;
  ...
```

2. Limited ability to extend design environment at runtime.

**Important:**

If you build application without runtime packages add **EC\_NO\_BPL**, **NEWNEWDESIGNER** defines to project "Conditional defines" (dialog "Project Options", tab sheet "Directories/Conditionals").

---

## 1.5 Integration with scripters

The main problem of integration designer environment and script engine is the ability to create and maintain event handlers. EControl Form Designer Pro contains special set of events and included in library method property editor which allow easy and event handlers creation and navigation in the same manner as we have in Delphi IDE.

Associations between event (procedural property) and script procedure may be saved in TStrings object as:

<ObjectName>.<EventPropertyName>=<ScriptProcedureName>. It is the easiest way to store associations.

Set StoreEvents (☑ see page 191) property to True to handle events by designer. In this case event associations will be saved in Events property (☑ see page 189) and designer's I/O functions will save these events in file or stream automatically.

**TzFormDesigner events for managing procedural properties:**

**OnGetScriptProc** (☑ see page 199) - called to get name of the script procedure assigned to the particular event of the object. Use this event handler for manual events processing.

**OnSetScriptProc** (☑ see page 200) - called to assign script procedure to the particular event of the object. If the script procedure with the given name does not exist you should add it to the code editor. It occurs when user in the object inspector enters name, selects procedure in drop down list or double click on the empty event.

```
procedure TForm4.zFormDesigner1SetScriptProc(Sender, Instance: TObject;
  pInfo: PPropInfo; const EventProc: String);
begin
```

```

if EventProc <> '' then
begin
    // Creating event handler text body in code editor
    // .....
end;
end;

```

**OnShowMethod** (see page 201) - called when in the object inspector user double clicks on the event with assigned script procedure. Example below shows how to find and highlight script procedure using parsing results of EControl TSyntaxMemo.

```

// Show script procedure in code editor
function TForm4.ShowMethod(const MethodName: string): Boolean;
var R: TTextRange;
    st, en: integer;
begin
    R := FindMethod(MethodName);
    Result := R <> nil;
    if Result then
        with CodeEditor.SyntObj do
        begin
            st := Tags[R.StartIdx].StartPos;
            if R.EndIdx <> -1 then
                en := Tags[R.EndIdx].EndPos
            else
                en := st;
            Windows.SetFocus(CodeEditor.Handle);
            CodeEditor.CaretStrPos := st;
            CodeEditor.CaretStrPos := en;
            CodeEditor.SetSelection(st, en - st);
        end;
    end;

    // Check existing script procedure
    function TForm4.FindMethod(const MethName: string): TTextRange;
    var i: integer;
        R: TTagBlockCondition;
    begin
        if MethName <> '' then
            with CodeEditor.SyntObj do
            begin
                R := TTagBlockCondition(Owner.BlockRules.ItemByName('function'));
                if R <> nil then
                    for i := 0 to RangeCount - 1 do
                        if (Ranges[i].Rule = R) and
                            SameText(TagStr[Ranges[i].StartIdx + 1], MethName) then
                            begin
                                Result := Ranges[i];
                                Exit;
                            end;
                    end;
                end;
            end;
            Result := nil;
        end;
    end;

```

**OnGetMethodNames** (see page 198) - called when user drop down pop-up list with events in the object inspector. Example below shows how to extract script procedure names in code editor using results of text analysis by the EControl Syntax Editor SDK.

```

// Returns existed script procedures
// Syntax Analyzer is used to extract available functions
// this allows correc event processing if the code contains erros
procedure TForm4.zFormDesigner1GetMethodNames(Sender: TObject;
    TypeData: PTypeData; Proc: TGetStrProc);
var i: integer;
    R: TTagBlockCondition;
begin
    with CodeEditor.SyntObj do
    begin
        // Looking for all text ranges with rule "function"
        R := TTagBlockCondition(Owner.BlockRules.ItemByName('function'));
        if R <> nil then
            for i := 0 to RangeCount - 1 do
                if (Ranges[i].Rule = R) then

```

```

        Proc(TagStr[Ranges[i].StartIdx + 1]);
    end;
end;

```

Using text analysis provided by EControl Syntax Editor SDK allows you to make language independent text processing (for example, to make this demo working with other script language you should only have a rule "function" in the lexer of that language).

Script source (TStrings) and events associations (TStrings) may be stored with form which may be saved to any storage (file - text or binary resource, database - in BLOB or MEMO field, etc.).

Example of form declaration with scripting support:

```

TScriptForm = class(TForm)
...
published
    property Code: TStrings ...
    property Events: TStrings ...
end;

```

Using such approach such form may be an independent all-sufficient unit.

The best solution is to combine EControl Form Designer Pro and EControl Syntax Editor SDK to develop convenient design environment in your applications!

## 1.6 License

### EControl Form Designer Pro

---

Copyright (c) 2004 - 2011, EControl Ltd., Zakharov Michael, All Rights Reserved

All copyrights to EControl Form Designer Pro are exclusively owned by the author - Zakharov Michael.

This software and accompanying documentation are protected by Russian Federation copyright law and also by International Treaty provisions. Any use of this software in violation of copyright law or the terms of this agreement will be prosecuted to the best of our ability.

### License Agreement

---

You should carefully read the following terms and conditions before using the software. By using this software you indicate that you accept the present license agreement.

### Registered Version (Single Developer License)

---

One registered copy of this software may either be used by a single developer who uses the software personally on one or more computers, or installed on a single workstation used non-simultaneously by multiple developers, but not both. Duration of support service is 1 year. Support service includes: free updates of the library and documentation, priority processing of

requested features and fixes, direct e-mail support from author.

#### **Registered Version (Site License)**

-----

One registered copy of this software may be used by unlimited number of developers within the company and may be installed on any number of computers in the company. Duration of support service is 2 year. Support service includes: free updates of the library and documentation, priority processing of requested features and fixes, direct support from author.

#### **Shareware Version**

-----

You may use shareware version for 30 days (TRIAL period). After that period has ended, to continue using this product you have to purchase the Registered version.

#### **Distribution**

-----

Provided that you verify that you are distributing the Shareware Version you are hereby licensed to make as many copies of the Shareware version of the software and the documentation as you wish; give exact copies of the original Shareware version to anyone; and distribute the Shareware version of the software and the documentation in its unmodified form via electronic means. There is no charge for any of the above-mentioned actions.

You are prohibited from charging, or requesting donations, for any such copies, however made; and from distributing the software and / or the documentation with other products (commercial or otherwise) without a prior written permission

#### **DISCLAIMER**

-----

This software is provided on an "as is" basis without warranty of any kind, expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. The person using the software bears all risk as to the quality and performance of the software. The author will not be liable for any special, incidental, consequential, indirect or similar damages due to loss of data or any other reason, even if the author or an agent of the author has been advised of the possibility of such damages. In no event shall the author's liability for any damages ever exceed the price paid for the license to use the software, regardless of the form of the claim.

#### **Registration**

-----

Ordering and registration of this software are available at:

<http://www.econtrol.ru/order.html>

---

## 1.7 Examples

## 1.7.1 TBtnMargins.Margins example

This example demonstrates how to set Margins property at run-time

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  PalettePanel1.Margins.Left      := 5;
  PalettePanel1.Margins.Top       := 2;
  PalettePanel1.Margins.Bottom    := 2;
  PalettePanel1.Margins.Right     := 5;
  PalettePanel1.Margins.BtnHorz   := 4;
  PalettePanel1.Margins.BtnVert   := 10;
end;
```

## 1.7.2 ButtonClick Example

This example demonstrates how descendant of TCustomBtnPanel overrides ButtonClick method to assign current class type in DsnManager

```
procedure TPalettePanel.ButtonClick(AButton: integer; Shift: TShiftState);
begin
  DsnManager.ComponentClass := TComponentClassInfo(FCmpList[AButton]).AClass;
  DsnManager.MultiCreate := ssShift in Shift;
  inherited;
end;
```

## 1.7.3 Registration Method Example

This example demonstrates how to register methods and event handlers

### Variant 1. Using MethRegister.AddMethod function

Step 1.

Create event handler or appropriate method (with right signature),

for example

```
procedure TForm1.User_ListViewCustomDraw(Sender: TCustomListView;
  const ARect: TRect; var DefaultDraw: Boolean);
begin
  //
end;

procedure TForm1.AnotherCustomDraw(Sender: TCustomListView;
  const ARect: TRect; var DefaultDraw: Boolean);
begin
```

#### Attention

It is necessary to place those declarations in **published** or **'automatic'** (where all form's field and methods is automatically added) section.

Step 2.

Somewhere in initialization code place MethRegister.AddMethod's calls

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    ...
    MethRegister.AddMethod(GetTypeData(TypeInfo(TLVCustomDrawEvent)),
        Addr(TForm1.User_ListViewCustomDraw), Self);
    MethRegister.AddMethod(GetTypeData(TypeInfo(TLVCustomDrawEvent)),
        Addr(TForm1.AnotherCustomDraw), Self);
    ...
end;

```

## Variant 2. Using Helper-functions

You can create helper functions for simple reuse of the same methods (with equal signature)

Step 1.

Create helper function for TLVCustomDrawEvent type

```

procedure AddCustomDrawEvent_Helper(ev: TLVCustomDrawEvent);
begin
    MethRegister.AddMethod(GetTypeData(TypeInfo(TLVCustomDrawEvent)), PMethod(@@ev)^);
end;

```

Step 2.

Now you can register method with more intuitive way

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    ...
    AddCustomDrawEvent_Helper(User_ListViewCustomDraw);
    AddCustomDrawEvent_Helper(AnotherCustomDraw);
    ...
end;

```

## Notes

MethRegister object have a list of built-in helpers

- procedure AddNotifyEvent(ev: TNotifyEvent);
- procedure AddMouseEvent(ev: TMouseEvent);
- procedure AddMouseMoveEvent(ev: TMouseMoveEvent);
- procedure AddKeyEvent(ev: TKeyEvent);
- procedure AddKeyPressEvent(ev: TKeyPressEvent);
- procedure AddDragOverEvent(ev: TDragOverEvent);
- procedure AddDragDropEvent(ev: TDragDropEvent);
- procedure AddStartDragEvent(ev: TStartDragEvent);
- procedure AddEndDragEvent(ev: TEndDragEvent);
- procedure AddDockDropEvent(ev: TDockDropEvent);
- procedure AddDockOverEvent(ev: TDockOverEvent);
- procedure AddUnDockEvent(ev: TUnDockEvent);
- procedure AddStartDockEvent(ev: TStartDockEvent);
- procedure AddGetSiteInfoEvent(ev: TGetSiteInfoEvent);
- procedure AddCanResizeEvent(ev: TCanResizeEvent);
- procedure AddConstrainedResizeEvent(ev: TConstrainedResizeEvent);
- procedure AddMouseWheelEvent(ev: TMouseWheelEvent);
- procedure AddMouseWheelUpDownEvent(ev: TMouseWheelUpDownEvent);



- procedure AddContextPopupEvent(ev: TContextPopupEvent);

---

## 1.8 Change Log

---

### 1.8.1 Version 2.00

#### Version 2.0 (from version 1.5)

##### ***ecDIList.pas***

Custom property list control implemented using classes: TCustomPropList (see page 46), TPropertyItem (see page 61), TPropListRoot (see page 65). Some functionality of inspector list is moved to these classes.

##### ***ecExtEdit.pas***

TUnicodeEdit (see page 107) - base edit control which is used as base class of property edit control. This control may be used as Ansi and as Unicode control.

TBtnEdit (see page 70) is derived from TUnicodeEdit. It contains new status area at left side of edit control. This status area is used to draw status image in property edit control.

TExEdit class renamed on TCustomEditEx (see page 78).

##### ***ecToolList.pas (new)***

TToolList (see page 125) control - categorized items list, with functionality similar Delphi's tools palette control. It is used as base class for TPaletteToolList. (see page 470)

##### ***ecHintHelper.pas (new)***

Provides unified hint processing. Hint helpers used by the tool lists, button panels and inspector lists, i.e.

TCustomBtnPanel.HintProps Property (see page 26)

TBtnPanel.HintProps Property (see page 37)

TPalettePanel.HintProps Property (see page 371)

TPaletteTab.HintProps Property (see page 382)

TCustomToolList.HintProps Property (see page 121)

TToolList.HintProps Property (see page 134)

TPaletteToolList.HintProps Property (see page 482)

TCustomInspectorList.HintProps Property (see page 410)

***ed\_DsnBase.pas***

Handling drag&drop operations:

TBaseDesigner.DragDrop Method (see page 147)

TBaseDesigner.DragOver Method (see page 148)

TBaseDesigner.OnDragDrop Event (see page 152)

TBaseDesigner.OnDragOver Event (see page 152)

***ed\_Designer.pas***

New functionality:

TzCustomFormDesigner.AddCompEditorMenu Method (see page 170)

TzCustomFormDesigner.CheckAction Method (see page 171)

TzCustomFormDesigner.ClearCompEditorMenu Method (see page 171)

TzCustomFormDesigner.EditAction Method (see page 172)

TzCustomFormDesigner.GetEditState Method (see page 173)

TzCustomFormDesigner.IsLocked Method (see page 179)

TzCustomFormDesigner.SelectedComponent Method (see page 185)

TzCustomFormDesigner.ShowPopupMenu Method (see page 186)

TzCustomFormDesigner.UndoLoad Property (see page 192)

TzCustomFormDesigner.OnCanEdit Event (see page 194)

TzCustomFormDesigner.OnGetComponentLocked Event (see page 195)

TzCustomFormDesigner.OnGetObjectName Event (see page 195)

***ed\_RegComps.pas***

TComponentClassInfo Class (see page 242) was redesigned, fields are moved to private section. Property AClass was replaced with property TComponentClassInfo.ComponentClass (see page 244). New Unicode property TComponentClassInfo.DisplayName (see page 244) used by the component palette and palette tool list.

Registered component information may be accessed using TPackageMng.ComponentCount (see page 255) and TPackageMng.Components (see page 255). Direct access via property *CompInfos* is removed.

Added several functions to manage component palette.

See TPackageMng class (see page 248).

***edcCmbCombo.pas***

New functionality to manage drop-down list:

TComponentCombo.DoAddObject Method (🔗 see page 345)

TComponentCombo.OnCanAddObject Event (🔗 see page 358)

Automatic hints:

TComponentCombo.AutoHint Property (🔗 see page 347)

### ***edcPropCtrl.pas***

New functionality of inspector list:

TCustomInspectorList.PopupListAlign (🔗 see page 411) - alignment of popup list.

TCustomInspectorList.OnChangeSelection (🔗 see page 412) - occurs when selected objects are changing.

TCustomInspectorList.OnGetPropReadOnly (🔗 see page 413) - occurs to define whether property is read-only.

TCustomInspectorList.OnSetPropValueA (🔗 see page 413) ; TCustomInspectorList.OnSetPropValueW (🔗 see page 413) - allows adjusting value to be written in property.

TCustomInspectorList.DefPropNameDraw (🔗 see page 409) - disables property name drawing by the property editors.

### ***edcPropEdit.pas***

Uses new interface to draw and process status images:

IPropertyStatusImage Interface (🔗 see page 464)

New functionality to control value updating:

TCustomPropertyEdit.ChangePropertyValue Method (🔗 see page 453)

TCustomPropertyEdit.OnSetPropValueA Event (🔗 see page 457)

TCustomPropertyEdit.OnSetPropValueW Event (🔗 see page 457)

### ***eddObjInspFrm.pas (new)***

Object inspector frame control. Utilizes object inspector functionality for fast integration in application. See TObjectInspectorFrame Class (🔗 see page 495) for more details.

---

## **1.8.2 Version 2.10**

Version 2.1 (from version 2.0 (🔗 see page 12))

CodeGear RAD Studio 2009 support.

**ed\_Designer.pas**

TzCustomFormDesigner.SelectedComponentsCount Method (🔗 see page 180)

TzCustomFormDesigner.OnPopUndo Event (🔗 see page 196)

TzCustomFormDesigner.OnPushUndo Event (🔗 see page 197)

TzCustomFormDesigner.OnSetNewName Event (🔗 see page 197)

TCustomFormDesigner supports standard actions TEditCopy, TEditCut, TEditPaste, TEditUndo, TEditSelectAll, TEditDelete and allows handle other standard actions using corresponding events:

TzCustomFormDesigner.OnExecuteAction Event (🔗 see page 195)

TzCustomFormDesigner.OnUpdateAction Event (🔗 see page 198)

### **edcCmbCombo.pas**

TComponentCombo.OnGetClassName Event (🔗 see page 358)

### **edManager.pas**

TDesignerManager.SetActiveDesigner Method (🔗 see page 510)

See this topic for sample of using multiple groups of designers (each group with own manager).

### **ecToolList.pas**

TCustomToolList.RightClickSelect Property (🔗 see page 122)

TCustomToolList.Selected Property (🔗 see page 122)

TCustomToolList.OnItemArranged Event (🔗 see page 123)

---

## **1.8.3 Version 2.20**

### **Version 2.2 (from version 2.1 (🔗 see page 14))**

Main changes:

1. Extended events support (in resource files)
2. Extended guidelines styles
3. Group operations
4. Local/global markers
5. Parent dragging limitation (ON/OFF)
6. Operations in inactive mode (ability of using component combo, object tree, object inspector with inactive designer)
7. Other fixes and minor improvements.

### **ed\_Designer.pas**

Simple file or stream I/O

TzCustomFormDesigner.SaveToFile Method (🔗 see page 179)

TzCustomFormDesigner.SaveToStream Method (🔗 see page 180)

TzCustomFormDesigner.LoadFromFile Method (see page 177)

TzCustomFormDesigner.LoadFromStream Method (see page 178)

TzCustomFormDesigner.IgnoreReadErrors Property (see page 191)

Self storage and events processing:

TzCustomFormDesigner.Events Property (see page 189)

TzCustomFormDesigner.StoreEvents Property (see page 191)

TzCustomFormDesigner.DragParentLimit Property (see page 189) - specifies whether drag mouse movement should be clipped by parent's client area.

TzCustomFormDesigner.GuidelinesStyle Property (see page 191) - specifies guidelines options. Ability to display guidelines in static mode, in keyboard operations.

TzCustomFormDesigner.Groups Property (see page 190) - component grouping support.

TControlGroups Class (see page 156) - control groups class.

### **ed\_dsncont.pas**

TDesignSurface.RulerClientArea Property (see page 240) - specifies whether ruler displays scale only for client area.

### **ed\_ObjTree.pas**

TCustomDesignerObjTree.Designer Property (see page 271) - Allows direct linking to the particular designer. In this case object tree will work with objects of the Designer.Root and will work even when designer is not active.

### **edActns.pas**

TdsnGroupControls Class (see page 323) - Group selected controls action.

TdsnUngroupControls Class (see page 339) - Ungroup selected controls action.

### **edcCmbCombo.pas**

TComponentCombo.Designer Property (see page 348) - Allows direct linking to the particular designer. Component combo can work also with inactive designer.

### **edIOUtils.pas**

Updated functions to support inline events saving.

zReadCmpFromFile (see page 504)

zReadCmpFromStream (see page 504)

zWriteCmpToFile (🔗 see page 505)

zWriteCmpToStream (🔗 see page 505)

### eduServObj.pas

TzBoundCtrl.Local Property (🔗 see page 522) - specifies whether markers are visible only in the parent window of selected control. In Code Gear designer - markers are local. In MS Stidio designer - markers are not local.

---

## 1.8.4 Version 2.30

### Version 2.3 (from version 2. (🔗 see page 14)2)

1. Improved design grid painting (faster)
2. Customization of Tool Component Palette, see TPaletteToolList.CustomItems Property (🔗 see page 480)
3. Corrected inspector painting
4. Fixed problem of object inspector under D2009 (when properties of type string were not shown in object editor frame). Added property TInspectorList.TypeSelector (🔗 see page 412).
5. Fixed problem of incorrect images deduplication procedure when adding actions in action list editor.
6. Registration of standard actions (StdRegComps.pas)
7. Redesigned standard actions (cut, copy, paste, delete, undo, select all). Now they derived from TEdit\_\_\_\_\_ actions. This allows using them for edit controls and for designer.
8. Property search functionality. In object inspector press TAB and input first letters of property display name to search it in list. See TCustomInspectorList.SearchPropMode Property (🔗 see page 411), TCustomInspectorList.SearchPropKey Property (🔗 see page 411).
9. Using drag images when dragging component from component palette or component tool list. See TPalettePanel.DragImageType Property (🔗 see page 370), TPaletteToolList.DragImageType Property (🔗 see page 481), TPaletteTab.PalettePanel Property (🔗 see page 386)
10. TzCustomFormDesigner.OnDrawControl Event (🔗 see page 195)

---

## 1.8.5 Version 2.40

### Version 2.4 (from version 2. (🔗 see page 14)3)

1. RAD Studio XE support.
2. "Show Tab Order" design mode - allows easy to view and change tab order (TzCustomFormDesigner.TabOrderIcons Property (🔗 see page 191)).
3. RTL support. Allows designing of windows with WS\_EX\_LAYOUTRTL style (right-to-left).

---

## 1.8.6 Version 2.50

Embarcadero RAD Studio XE2 support.

# 1.9 ecBtnPanel Namespace

## 1.9.1 Classes

The following table lists classes in this documentation.

### Classes

Class	Description
TBtnMargins (🔗 see page 18)	Contains margins and spaces for a button's array in TCustomBtnPanel (🔗 see page 20)
TCustomBtnPanel (🔗 see page 20)	TCustomBtnPanel is the base class for panels with button array.
TBtnPanel (🔗 see page 28)	Control with multiple buttons.

### 1.9.1.1 TBtnMargins Class

Contains margins and spaces for a button's array in TCustomBtnPanel (🔗 see page 20)

#### Class Hierarchy



```
TBtnMargins = class(TPersistent);
```

#### File

ecBtnPanel

#### Description

TBtnMargins is used in the Margins property of TCustomBtnPanel (🔗 see page 20) and its descendants (i.e. TPalettePanel). It specifies the Left (🔗 see page 19), Top (🔗 see page 20), Right (🔗 see page 20) and Bottom (🔗 see page 19) margins between buttons and underlaying panel as well as horizontal and vertical spaces between buttons themselves.

#### Example

This example (🔗 see page 10) demonstrates how to set Margins property at run-time

#### Members







##### TBtnMargins Properties

TBtnMargins Properties	Description
🔗 Bottom (🔗 see page 19)	Specifies the bottom margin between underlaying panel and the very lower line of buttons
🔗 BtnHorz (🔗 see page 19)	Specifies the horizontal spaces between the buttons
🔗 BtnVert (🔗 see page 19)	Specifies the vertical spaces between the buttons
🔗 Left (🔗 see page 19)	Specifies the left margin between underlaying panel and the very left button.
🔗 Right (🔗 see page 20)	Specifies the right margin between underlaying panel and the very right button.
🔗 Top (🔗 see page 20)	Specifies the top margin between underlaying panel and the very upper line of buttons

#### Legend

🔗	Property
---	----------

**TBtnMargins Properties**

TBtnMargins Properties	Description
 Bottom (see page 19)	Specifies the bottom margin between underlaying panel and the very lower line of buttons
 BtnHorz (see page 19)	Specifies the horizontal spaces between the buttons
 BtnVert (see page 19)	Specifies the vertical spaces between the buttons
 Left (see page 19)	Specifies the left margin between underlaying panel and the very left button.
 Right (see page 20)	Specifies the right margin between underlaying panel and the very right button.
 Top (see page 20)	Specifies the top margin between underlaying panel and the very upper line of buttons

**1.9.1.1.1 TBtnMargins Properties****1.9.1.1.1.1 TBtnMargins.Bottom Property**

Specifies the bottom margin between underlaying panel and the very lower line of buttons

```
property Bottom: integer;
```

**Description**

Use the Bottom property to determine or set bottom margin.

**Example**

This example (see page 10) demonstrates how to set Bottom property at run-time

**1.9.1.1.1.2 TBtnMargins.BtnHorz Property**

Specifies the horizontal spaces between the buttons

```
property BtnHorz: integer;
```

**Description**

Use the BtnHorz property to determine or set horizontal spaces between buttons on the panel.

**Example**

This example (see page 10) demonstrates how to set BtnHorz property at run-time

**1.9.1.1.1.3 TBtnMargins.BtnVert Property**

Specifies the vertical spaces between the buttons

```
property BtnVert: integer;
```

**Description**

Use the BtnVert property to determine or set vertical spaces between buttons on the panel.

**Example**

This example (see page 10) demonstrates how to set BtnVert property at run-time

**1.9.1.1.1.4 TBtnMargins.Left Property**

Specifies the left margin between underlaying panel and the very left button.

```
property Left: integer;
```

**Description**

Use the Left property to determine or set the left margin.

**Example**

This example (see page 10) demonstrates how to set Left property at run-time



### 1.9.1.1.1.5 TBtnMargins.Right Property

Specifies the right margin between underlaying panel and the very right button.

```
property Right: integer;
```

#### Description

Use the Right property to determine or set the right margin.

#### Example

This example (see page 10) demonstrates how to set Right property at run-time

### 1.9.1.1.1.6 TBtnMargins.Top Property

Specifies the top margin between underlaying panel and the very upper line of buttons

```
property Top: integer;
```

#### Description

Use the Top property to determine or set top margin.

#### Example

This example (see page 10) demonstrates how to set Top property at run-time

## 1.9.1.2 TCustomBtnPanel Class

TCustomBtnPanel is the base class for panels with button array.

#### Class Hierarchy



```
TCustomBtnPanel = class(TCustomPanel);
```

#### File

ecBtnPanel

#### Description













Buttons are not controls. This class was created to optimize rendering of big buttons array, for example, in component palette.

#### Members




#### TCustomBtnPanel Methods

TCustomBtnPanel Methods	Description
ButtonAtPos (see page 22)	Returns the index of the button indicated by the coordinates of a point on the panel.
ButtonClick (see page 22)	Generates an OnButtonClick (see page 27) event.
ButtonRect (see page 22)	Indicates the rectangle occupied by the particular button.
CanAutoSize (see page 23)	Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents.
Create (see page 23)	Creates and initializes a TCustomBtnPanel instance.
Destroy (see page 23)	Destroys an instance of TCustomBtnPanel.
DrawButton (see page 23)	Generates an OnDrawButton (see page 27) event.
GetButtonHint (see page 24)	Returns hint text for the particular button.
InvalidateButtons (see page 24)	Repaints just pushed and released buttons.
Loaded (see page 24)	Finally initializes the TCustomBtnPanel after it is loaded from a stream.
MouseDown (see page 24)	Calls ButtonClick (see page 22) method.
Paint (see page 24)	Renders the image of a TCustomBtnPanel.






**TCustomBtnPanel Properties**

TCustomBtnPanel Properties	Description
 AutoSize (see page 25)	Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 ButtonCount (see page 25)	Indicates the number of buttons in the array.
 ButtonHeight (see page 25)	Specifies the height of the buttons in array.
 ButtonWidth (see page 25)	Specifies the width of the buttons in array.
 Caption (see page 25)	Specifies a text string that identifies the control to the user. This is inherited property from TControl.
 DownButton (see page 26)	Specifies index of pressed button.
 Flat (see page 26)	Dictates whether the button should have a 2D look instead of the usual 3D look.
 HintProps (see page 26)	Provide properties to adjust hints processing.
 Margins (see page 26)	Specifies positioning of button array
 Orientation (see page 26)	Specifies whether the panel's array of button is horizontal or vertical.
 RowCount (see page 27)	Indicates the number of button rows in the panel.
 Transparent (see page 27)	Specifies whether the background of the button is transparent.




**TCustomBtnPanel Events**

TCustomBtnPanel Events	Description
 OnButtonClick (see page 27)	Occurs when the user clicks the button on the underlying panel.
 OnDrawButton (see page 27)	Occurs when a particular button on the panel needs to be drawn.
 OnGetButtonHint (see page 28)	Occurs before hint window will be displayed.






















**Legend**

	Method
	protected
	virtual
	Property
	Event




**TCustomBtnPanel Events**










TCustomBtnPanel Events	Description
 OnButtonClick (see page 27)	Occurs when the user clicks the button on the underlying panel.
 OnDrawButton (see page 27)	Occurs when a particular button on the panel needs to be drawn.
 OnGetButtonHint (see page 28)	Occurs before hint window will be displayed.

**TCustomBtnPanel Methods**

TCustomBtnPanel Methods	Description
 ButtonAtPos (see page 22)	Returns the index of the button indicated by the coordinates of a point on the panel.
  ButtonClick (see page 22)	Generates an OnButtonClick (see page 27) event.
 ButtonRect (see page 22)	Indicates the rectangle occupied by the particular button.
  CanAutoSize (see page 23)	Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents.
  Create (see page 23)	Creates and initializes a TCustomBtnPanel instance.
  Destroy (see page 23)	Destroys an instance of TCustomBtnPanel.
  DrawButton (see page 23)	Generates an OnDrawButton (see page 27) event.
  GetButtonHint (see page 24)	Returns hint text for the particular button.
 InvalidateButtons (see page 24)	Repaints just pushed and released buttons.
  Loaded (see page 24)	Finally initializes the TCustomBtnPanel after it is loaded from a stream.
  MouseDown (see page 24)	Calls ButtonClick (see page 22) method.
  Paint (see page 24)	Renders the image of a TCustomBtnPanel.

**TCustomBtnPanel Properties**

TCustomBtnPanel Properties	Description
 AutoSize (see page 25)	Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 ButtonCount (see page 25)	Indicates the number of buttons in the array.
 ButtonHeight (see page 25)	Specifies the height of the buttons in array.

 ButtonWidth (see page 25)	Specifies the width of the buttons in array.
 Caption (see page 25)	Specifies a text string that identifies the control to the user. This is inherited property from TControl.
 DownButton (see page 26)	Specifies index of pressed button.
 Flat (see page 26)	Dictates whether the button should have a 2D look instead of the usual 3D look.
 HintProps (see page 26)	Provide properties to adjust hints processing.
 Margins (see page 26)	Specifies positioning of button array
 Orientation (see page 26)	Specifies whether the panel's array of button is horizontal or vertical.
 RowCount (see page 27)	Indicates the number of button rows in the panel.
 Transparent (see page 27)	Specifies whether the background of the button is transparent.

### 1.9.1.2.1 TCustomBtnPanel Methods

#### 1.9.1.2.1.1 TCustomBtnPanel.ButtonAtPos Method

Returns the index of the button indicated by the coordinates of a point on the panel.

```
function ButtonAtPos(Pos: TPoint): integer;
```

##### Description

Use ButtonAtPos to detect if a button exists at a particular point in the underlying panel.

The Pos parameter is the point in the panel in window coordinates.

If Pos is out of buttons coordinates, ButtonAtPos returns -1, otherwise it returns absolute index of underlying button.

ButtonAtPos is used internally for TCustomBtnPanel (see page 20) purposes, i.e. in TCustomBtnPanel.MouseDown (see page 24) and TCustomBtnPanel.WMMouseMove procedures to determine particular button for subsequent operations.

#### 1.9.1.2.1.2 TCustomBtnPanel.ButtonClick Method

Generates an OnButtonClick (see page 27) event.

```
procedure ButtonClick(AButton: integer; Shift: TShiftState); virtual;
```

##### Description

ButtonClick is called automatically when user presses mouse button with the mouse pointer over the particular button. Then it generates an OnButtonClick (see page 27) event.

The AButton parameter indicates index of corresponding button.

The Shift parameter indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button.

Override this method to add class-specific processing when the button clicks.

##### Example

See (see page 10) how descendant of TCustomBtnPanel (see page 20) overrides this method to assign current class type in DsnManager

#### 1.9.1.2.1.3 TCustomBtnPanel.ButtonRect Method

Indicates the rectangle occupied by the particular button.

```
function ButtonRect(idx: integer): TRect;
```

**Description**

Used internally to determine rectangle occupied by the particular button.

**1.9.1.2.1.4 TCustomBtnPanel.CanAutoSize Method**

Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents.

```
function CanAutoSize(var NewWidth: Integer; var NewHeight: Integer): Boolean; override;
```

**Description**

Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents. This is setter for the AutoSize (see page 25) property.

CanAutoSize is called automatically when the AutoSize (see page 25) property is true and an attempt is made to resize the control.

It resize control depending on RowCount (see page 27), ButtonCount (see page 25) and Orientation (see page 26) properties.

Method always returns True and the NewWidth and NewHeight parameters indicate the proposed new dimensions of the control.

**1.9.1.2.1.5 TCustomBtnPanel.Create Constructor**

Creates and initializes a TCustomBtnPanel instance.

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

**Description**

Use Create to programmatically instantiate a TCustomBtnPanel component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

**1.9.1.2.1.6 TCustomBtnPanel.Destroy Destructor**

Destroys an instance of TCustomBtnPanel.

```
destructor Destroy; override;
```

**Description**

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not nil before calling Destroy.

**1.9.1.2.1.7 TCustomBtnPanel.DrawButton Method**

Generates an OnDrawButton (see page 27) event.

```
procedure DrawButton(var Rect: TRect; Index: Integer); virtual;
```

**Description**

Simply generates an OnDrawButton (see page 27) event after rendering this button in TCustomBtnPanel.Paint (see page 24).

The Rect parameter indicates the location of the button on the canvas.

The Index parameter indicates index of corresponding button.

#### 1.9.1.2.1.8 TCustomBtnPanel.GetButtonHint Method

Returns hint text for the particular button.

```
function GetButtonHint(Index: Integer): WideString; virtual;
```

##### Description

Returns hint text for the particular button.

The Index parameter indicates index of corresponding button.

#### 1.9.1.2.1.9 TCustomBtnPanel.InvalidateButtons Method

Repaints just pushed and released buttons.

```
procedure InvalidateButtons(b1: integer; b2: integer);
```

##### Description

Completely repaints just pushed and released buttons.

Used internally immediately after setting new button down.

First parameter indicates released, second one - pushed button.

#### 1.9.1.2.1.10 TCustomBtnPanel.Loaded Method

Finally initializes the TCustomBtnPanel (see page 20) after it is loaded from a stream.

```
procedure Loaded; override;
```

#### 1.9.1.2.1.11 TCustomBtnPanel.MouseDown Method

Calls ButtonClick (see page 22) method.

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
override;
```

##### Description

Determines button at cursor and calls ButtonClick (see page 22) method.

The Button parameter determines which mouse button the user pressed.

Shift indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button.

X and Y are the pixel coordinates of the mouse pointer within the client area of the TCustomBtnPanel (see page 20).

#### 1.9.1.2.1.12 TCustomBtnPanel.Paint Method

Renders the image of a TCustomBtnPanel (see page 20).

```
procedure Paint; override;
```

##### Description

Renders the whole image of a TCustomBtnPanel (see page 20) including buttons, images and so on.

### 1.9.1.2.2 TCustomBtnPanel Properties

#### 1.9.1.2.2.1 TCustomBtnPanel.AutoSize Property

Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents. This is inherited property from TControl.

```
property AutoSize;
```

##### Description

Use AutoSize to specify whether the TCustomBtnPanel (see page 20) sizes itself automatically. When AutoSize is True, the TCustomBtnPanel (see page 20) resizes automatically to arrange all the buttons in array.

By default, AutoSize is True.

#### 1.9.1.2.2.2 TCustomBtnPanel.ButtonCount Property

Indicates the number of buttons in the array.

```
property ButtonCount: integer;
```

##### Description

ButtonCount gives the number of buttons in the array. Set this property to change count of buttons.

##### Notes

Trying to set ButtonCount to negative number is ignored.

#### 1.9.1.2.2.3 TCustomBtnPanel.ButtonHeight Property

Specifies the height of the buttons in array.

```
property ButtonHeight: integer;
```

##### Description

ButtonHeight represents the height, in pixels, of the buttons on the panel.

Setting this property to value less then 2 pixels throws an Exception.

By default, ButtonHeight is 25.

#### 1.9.1.2.2.4 TCustomBtnPanel.ButtonWidth Property

Specifies the width of the buttons in array.

```
property ButtonWidth: integer;
```

##### Description

ButtonWidth represents the width, in pixels, of the buttons on the panel.

Setting this property to value less then 2 pixels throws an Exception.

By default, ButtonWidth is 25.

#### 1.9.1.2.2.5 TCustomBtnPanel.Caption Property

Specifies a text string that identifies the control to the user. This is inherited property from TControl.

```
property Caption;
```

**Description**

Use Caption to specify the text string that labels the control.

By default, Caption is empty.

**1.9.1.2.2.6 TCustomBtnPanel.DownButton Property**

Specifies index of pressed button.

```
property DownButton: integer;
```

**Description**

Specifies index of the button which is in selected state where 0 is the first button, 1 is the second and so on;

Set this property to change selected button in the array.

By default, DownButton is -1;

**1.9.1.2.2.7 TCustomBtnPanel.Flat Property**

Dictates whether the button should have a 2D look instead of the usual 3D look.

```
property Flat: Boolean;
```

**Description**

Set Flat to True if you want the button to display the button without the edge bevel that gives buttons a 3D look.

By default, Flat is True.

**1.9.1.2.2.8 TCustomBtnPanel.HintProps Property**

Provide properties to adjust hints processing.

```
property HintProps: TechHintHelper;
```

**1.9.1.2.2.9 TCustomBtnPanel.Margins Property**

Specifies positioning of button array

```
property Margins: TBtnMargins;
```

**Description**

Margins specifies the Left, Top, Right and Bottom margins between buttons and underlaying panel as well as horizontal and vertical spaces between buttons themselves.

**Example**

This example (see page 10) demonstrates how to set Margins property at run-time

**1.9.1.2.2.10 TCustomBtnPanel.Orientation Property**

Specifies whether the panel's array of button is horizontal or vertical.

```
property Orientation: TRowOrientation;
```

**Description**

Use Orientation to specify whether the the panel's array of button is horizontal or vertical.

By default, Orientation is roHorizontal.

#### 1.9.1.2.2.11 TCustomBtnPanel.RowCount Property

Indicates the number of button rows in the panel.

**property** RowCount: integer;

##### Description

Read RowCount to determine the number of rows in the button's array are placed.

Set RowCount to add or delete rows to rearrange buttons.

By default, RowCount is 1.

#### 1.9.1.2.2.12 TCustomBtnPanel.Transparent Property

Specifies whether the background of the button is transparent.

**property** Transparent: Boolean;

##### Description

Use Transparent to specify whether the background of the button is transparent.

### 1.9.1.2.3 TCustomBtnPanel Events

#### 1.9.1.2.3.1 TCustomBtnPanel.OnButtonClick Event

Occurs when the user clicks the button on the underlying panel.

**property** OnButtonClick: TButtonClickEvent;

##### Description

Use the OnButtonClick event handler to respond when the user clicks the button on the TCustomBtnPanel (see page 20).

OnButtonClick occurs when the user presses mouse button with the mouse pointer over the corresponding rectangle of the underlying TCustomBtnPanel (see page 20).

The Sender parameter is the object whose event handler is called.

The Index parameter indicates index of pressed button.

#### 1.9.1.2.3.2 TCustomBtnPanel.OnDrawButton Event

Occurs when a particular button on the panel needs to be drawn.

**property** OnDrawButton: TDrawButtonEvent;

##### Description

Write an OnDrawButton event handler to draw the particular button.

The Sender parameter is the object whose event handler is called.

The Rect parameter indicates the location of the button on the canvas.

The Index parameter indicates index of corresponding button button.



### 1.9.1.2.3.3 TCustomBtnPanel.OnGetButtonHint Event

Occurs before hint window will be displayed.

**property** OnGetButtonHint: TGetButtonHintEvent;

#### Description

Use this event to specify hint text for particular button.

The Sender parameter is the object whose event handler is called.

The Index parameter indicates index of corresponding button button.

The AHint parameter used to assign Hint to be displayed for particular button.

## 1.9.1.3 TBtnPanel Class

Control with multiple buttons.

#### Class Hierarchy



TBtnPanel = **class**(TCustomBtnPanel);

#### File

ecBtnPanel

#### Description

Buttons are not controls. This class was created to optimize rendering of big buttons array, for example, in component palette.









#### Members

##### TCustomBtnPanel Methods








































TCustomBtnPanel Methods	Description
ButtonAtPos (see page 22)	Returns the index of the button indicated by the coordinates of a point on the panel.
ButtonClick (see page 22)	Generates an OnButtonClick (see page 27) event.
ButtonRect (see page 22)	Indicates the rectangle occupied by the particular button.
CanAutoSize (see page 23)	Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents.
Create (see page 23)	Creates and initializes a TCustomBtnPanel instance.
Destroy (see page 23)	Destroys an instance of TCustomBtnPanel.
DrawButton (see page 23)	Generates an OnDrawButton (see page 27) event.
GetButtonHint (see page 24)	Returns hint text for the particular button.
InvalidateButtons (see page 24)	Repaints just pushed and released buttons.
Loaded (see page 24)	Finally initializes the TCustomBtnPanel (see page 20) after it is loaded from a stream.
MouseDown (see page 24)	Calls ButtonClick (see page 22) method.
Paint (see page 24)	Renders the image of a TCustomBtnPanel (see page 20).

















##### TCustomBtnPanel Properties

TCustomBtnPanel Properties	Description
AutoSize (see page 25)	Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents. This is inherited property from TControl.
ButtonCount (see page 25)	Indicates the number of buttons in the array.
ButtonHeight (see page 25)	Specifies the height of the buttons in array.
ButtonWidth (see page 25)	Specifies the width of the buttons in array.




 Caption (see page 25)	Specifies a text string that identifies the control to the user. This is inherited property from TControl.
 DownButton (see page 26)	Specifies index of pressed button.
 Flat (see page 26)	Dictates whether the button should have a 2D look instead of the usual 3D look.
 HintProps (see page 26)	Provide properties to adjust hints processing.
 Margins (see page 26)	Specifies positioning of button array
 Orientation (see page 26)	Specifies whether the panel's array of button is horizontal or vertical.
 RowCount (see page 27)	Indicates the number of button rows in the panel.
 Transparent (see page 27)	Specifies whether the background of the button is transparent.

## TBtnPanel Class






TBtnPanel Class	Description
 Align (see page 32)	Determines how the control aligns within its container (parent control).
 Anchors (see page 33)	Specifies how the control is anchored to its parent.
 AutoSize (see page 33)	Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 BevelInner (see page 33)	Specifies the cut of the inner bevel.
 BevelOuter (see page 33)	Specifies the cut of the outer bevel.
 BevelWidth (see page 34)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
 BiDiMode (see page 34)	Specifies the bi-directional mode for the control.
 BorderStyle (see page 34)	Determines the style of the line drawn around the perimeter of the panel control.
 BorderWidth (see page 34)	Specifies the distance, in pixels, between the outer and inner bevels.
 ButtonCount (see page 35)	Indicates the number of buttons in the array.
 ButtonHeight (see page 35)	Specifies the height of the buttons in array.
 ButtonWidth (see page 35)	Specifies the width of the buttons in array.
 Color (see page 35)	Specifies the background color of the control.
 Constraints (see page 36)	Specifies the size constraints for the control.
 Ctl3D (see page 36)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DownButton (see page 36)	Specifies index of pressed button.
 DragCursor (see page 36)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 36)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 37)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 37)	Controls whether the control responds to mouse, keyboard, and timer events.
 Flat (see page 37)	Dictates whether the button should have a 2D look instead of the usual 3D look.
 Font (see page 37)	Controls the attributes of text written on or in the control.
 HintProps (see page 37)	Provide properties to adjust hints processing.
 Margins (see page 37)	Specifies positioning of button array
 OnButtonClick (see page 38)	Occurs when the user clicks the button on the underlying panel.
 OnCanResize (see page 38)	Occurs when an attempt is made to resize the control.
 OnClick (see page 38)	Occurs when the user clicks the control.
 OnConstrainedResize (see page 38)	Adjust resize constraints.
 OnContextPopup (see page 39)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 39)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 39)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 39)	Occurs when the user drags an object over a control.
 OnDrawButton (see page 40)	Occurs when a particular button on the panel needs to be drawn.
 OnEndDrag (see page 40)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 40)	Occurs when a control receives the input focus.
 OnExit (see page 40)	Occurs when the input focus shifts away from one control to another.
 OnGetButtonHint (see page 41)	Occurs before hint window will be displayed.
 OnMouseDown (see page 41)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 41)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

 OnMouseUp (see page 41)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnResize (see page 42)	Occurs immediately after the control is resized.
 OnStartDrag (see page 42)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 Orientation (see page 42)	Specifies whether the panel's array of button is horizontal or vertical.
 ParentBiDiMode (see page 42)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 42)	Determines where a control looks for its color information.
 ParentCtl3D (see page 43)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 43)	Determines where a control looks for its font information.
 ParentShowHint (see page 43)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 43)	Identifies the pop-up menu associated with the control.
 RowCount (see page 43)	Indicates the number of button rows in the panel.
 ShowHint (see page 43)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 TabOrder (see page 43)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 43)	Determines if the user can tab to a control.
 Transparent (see page 44)	Specifies whether the background of the button is transparent.
 Visible (see page 44)	Determines whether the component appears on screen.




## TCustomBtnPanel Events

TCustomBtnPanel Events	Description
 OnButtonClick (see page 27)	Occurs when the user clicks the button on the underlying panel.
 OnDrawButton (see page 27)	Occurs when a particular button on the panel needs to be drawn.
 OnGetButtonHint (see page 28)	Occurs before hint window will be displayed.




















## Legend

	Method
	protected
	virtual
	Property
	Event













## TCustomBtnPanel Events

TCustomBtnPanel Events	Description
 OnButtonClick (see page 27)	Occurs when the user clicks the button on the underlying panel.
 OnDrawButton (see page 27)	Occurs when a particular button on the panel needs to be drawn.
 OnGetButtonHint (see page 28)	Occurs before hint window will be displayed.

## TCustomBtnPanel Methods























TCustomBtnPanel Methods	Description
 ButtonAtPos (see page 22)	Returns the index of the button indicated by the coordinates of a point on the panel.
  ButtonClick (see page 22)	Generates an OnButtonClick (see page 27) event.
 ButtonRect (see page 22)	Indicates the rectangle occupied by the particular button.
  CanAutoSize (see page 23)	Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents.
 Create (see page 23)	Creates and initializes a TCustomBtnPanel instance.
 Destroy (see page 23)	Destroys an instance of TCustomBtnPanel.
  DrawButton (see page 23)	Generates an OnDrawButton (see page 27) event.
  GetButtonHint (see page 24)	Returns hint text for the particular button.
 InvalidateButtons (see page 24)	Repaints just pushed and released buttons.
  Loaded (see page 24)	Finally initializes the TCustomBtnPanel (see page 20) after it is loaded from a stream.
  MouseDown (see page 24)	Calls ButtonClick (see page 22) method.
  Paint (see page 24)	Renders the image of a TCustomBtnPanel (see page 20).

**TCustomBtnPanel Properties**

<b>TCustomBtnPanel Properties</b>	<b>Description</b>
 <b>AutoSize</b> ( <a href="#">see page 25</a> )	Specifies whether the TCustomBtnPanel ( <a href="#">see page 20</a> ) sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 <b>ButtonCount</b> ( <a href="#">see page 25</a> )	Indicates the number of buttons in the array.
 <b>ButtonHeight</b> ( <a href="#">see page 25</a> )	Specifies the height of the buttons in array.
 <b>ButtonWidth</b> ( <a href="#">see page 25</a> )	Specifies the width of the buttons in array.
 <b>Caption</b> ( <a href="#">see page 25</a> )	Specifies a text string that identifies the control to the user. This is inherited property from TControl.
 <b>DownButton</b> ( <a href="#">see page 26</a> )	Specifies index of pressed button.
 <b>Flat</b> ( <a href="#">see page 26</a> )	Dictates whether the button should have a 2D look instead of the usual 3D look.
 <b>HintProps</b> ( <a href="#">see page 26</a> )	Provide properties to adjust hints processing.
 <b>Margins</b> ( <a href="#">see page 26</a> )	Specifies positioning of button array
 <b>Orientation</b> ( <a href="#">see page 26</a> )	Specifies whether the panel's array of button is horizontal or vertical.
 <b>RowCount</b> ( <a href="#">see page 27</a> )	Indicates the number of button rows in the panel.
 <b>Transparent</b> ( <a href="#">see page 27</a> )	Specifies whether the background of the button is transparent.

**TBtnPanel Class**

<b>TBtnPanel Class</b>	<b>Description</b>
 <b>Align</b> ( <a href="#">see page 32</a> )	Determines how the control aligns within its container (parent control).
 <b>Anchors</b> ( <a href="#">see page 33</a> )	Specifies how the control is anchored to its parent.
 <b>AutoSize</b> ( <a href="#">see page 33</a> )	Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 <b>BevelInner</b> ( <a href="#">see page 33</a> )	Specifies the cut of the inner bevel.
 <b>BevelOuter</b> ( <a href="#">see page 33</a> )	Specifies the cut of the outer bevel.
 <b>BevelWidth</b> ( <a href="#">see page 34</a> )	Determines the width, in pixels, of both the inner and outer bevels of a panel.
 <b>BiDiMode</b> ( <a href="#">see page 34</a> )	Specifies the bi-directional mode for the control.
 <b>BorderStyle</b> ( <a href="#">see page 34</a> )	Determines the style of the line drawn around the perimeter of the panel control.
 <b>BorderWidth</b> ( <a href="#">see page 34</a> )	Specifies the distance, in pixels, between the outer and inner bevels.
 <b>ButtonCount</b> ( <a href="#">see page 35</a> )	Indicates the number of buttons in the array.
 <b>ButtonHeight</b> ( <a href="#">see page 35</a> )	Specifies the height of the buttons in array.
 <b>ButtonWidth</b> ( <a href="#">see page 35</a> )	Specifies the width of the buttons in array.
 <b>Color</b> ( <a href="#">see page 35</a> )	Specifies the background color of the control.
 <b>Constraints</b> ( <a href="#">see page 36</a> )	Specifies the size constraints for the control.
 <b>Ctl3D</b> ( <a href="#">see page 36</a> )	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 <b>DownButton</b> ( <a href="#">see page 36</a> )	Specifies index of pressed button.
 <b>DragCursor</b> ( <a href="#">see page 36</a> )	Indicates the image used to represent the mouse pointer when the control is being dragged.
 <b>DragKind</b> ( <a href="#">see page 36</a> )	Specifies whether the control is being dragged normally or for docking.
 <b>DragMode</b> ( <a href="#">see page 37</a> )	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 <b>Enabled</b> ( <a href="#">see page 37</a> )	Controls whether the control responds to mouse, keyboard, and timer events.
 <b>Flat</b> ( <a href="#">see page 37</a> )	Dictates whether the button should have a 2D look instead of the usual 3D look.
 <b>Font</b> ( <a href="#">see page 37</a> )	Controls the attributes of text written on or in the control.
 <b>HintProps</b> ( <a href="#">see page 37</a> )	Provide properties to adjust hints processing.
 <b>Margins</b> ( <a href="#">see page 37</a> )	Specifies positioning of button array
 <b>OnButtonClick</b> ( <a href="#">see page 38</a> )	Occurs when the user clicks the button on the underlying panel.
 <b>OnCanResize</b> ( <a href="#">see page 38</a> )	Occurs when an attempt is made to resize the control.
 <b>OnClick</b> ( <a href="#">see page 38</a> )	Occurs when the user clicks the control.
 <b>OnConstrainedResize</b> ( <a href="#">see page 38</a> )	Adjust resize constraints.
 <b>OnContextPopup</b> ( <a href="#">see page 39</a> )	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 <b>OnDbClick</b> ( <a href="#">see page 39</a> )	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 <b>OnDragDrop</b> ( <a href="#">see page 39</a> )	Occurs when the user drops an object being dragged.
 <b>OnDragOver</b> ( <a href="#">see page 39</a> )	Occurs when the user drags an object over a control.
 <b>OnDrawButton</b> ( <a href="#">see page 40</a> )	Occurs when a particular button on the panel needs to be drawn.

 OnEndDrag (see page 40)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 40)	Occurs when a control receives the input focus.
 OnExit (see page 40)	Occurs when the input focus shifts away from one control to another.
 OnGetButtonHint (see page 41)	Occurs before hint window will be displayed.
 OnMouseDown (see page 41)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 41)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 41)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnResize (see page 42)	Occurs immediately after the control is resized.
 OnStartDrag (see page 42)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 Orientation (see page 42)	Specifies whether the panel's array of button is horizontal or vertical.
 ParentBiDiMode (see page 42)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 42)	Determines where a control looks for its color information.
 ParentCtl3D (see page 43)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 43)	Determines where a control looks for its font information.
 ParentShowHint (see page 43)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 43)	Identifies the pop-up menu associated with the control.
 RowCount (see page 43)	Indicates the number of button rows in the panel.
 ShowHint (see page 43)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 TabOrder (see page 43)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 43)	Determines if the user can tab to a control.
 Transparent (see page 44)	Specifies whether the background of the button is transparent.
 Visible (see page 44)	Determines whether the component appears on screen.

### 1.9.1.3.1 TBtnPanel Properties

#### 1.9.1.3.1.1 TBtnPanel.Align Property

Determines how the control aligns within its container (parent control).

**property** Align;

##### Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to `alLeft`. The value of `alLeft` for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is `alNone`, which means a control remains where it is positioned on a form or panel.

**Tip:** If Align is set to `alClient`, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

**Note:** To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

#### 1.9.1.3.1.2 TBtnPanel.Anchors Property

Specifies how the control is anchored to its parent.

**property** Anchors;

##### Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

**Note:** If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

#### 1.9.1.3.1.3 TBtnPanel.AutoSize Property

Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents. This is inherited property from TControl.

**property** AutoSize;

##### Description

Use AutoSize to specify whether the TCustomBtnPanel sizes itself automatically. When AutoSize is True, the TCustomBtnPanel resizes automatically to arrange all the buttons in array.

By default, AutoSize is True.

#### 1.9.1.3.1.4 TBtnPanel.BevelInner Property

Specifies the cut of the inner bevel.

**property** BevelInner;

##### Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

#### 1.9.1.3.1.5 TBtnPanel.BevelOuter Property

Specifies the cut of the outer bevel.

**property** BevelOuter;

**Description**

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

**1.9.1.3.1.6 TBtnPanel.BevelWidth Property**

Determines the width, in pixels, of both the inner and outer bevels of a panel.

**property** BevelWidth;

**Description**

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the BevelInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

**1.9.1.3.1.7 TBtnPanel.BiDiMode Property**

Specifies the bi-directional mode for the control.

**property** BiDiMode;

**Description**

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

**1.9.1.3.1.8 TBtnPanel.BorderStyle Property**

Determines the style of the line drawn around the perimeter of the panel control.

**property** BorderStyle;

**Description**

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

**1.9.1.3.1.9 TBtnPanel.BorderWidth Property**

Specifies the distance, in pixels, between the outer and inner bevels.

**property** BorderWidth;

**Description**

Use `BorderWidth` to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if `BevelInner` is `bvNone`. If the `Alignment` property is not `taCenter`, the Caption will be aligned to the inner edge of the border. This edge is `BorderWidth` pixels in from the outer bevel if `BevelInner` is `bvNone`. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the `BorderStyle` property.

**1.9.1.3.1.10 TBtnPanel.ButtonCount Property**

Indicates the number of buttons in the array.

```
property ButtonCount: integer;
```

**Description**

`ButtonCount` gives the number of buttons in the array. Set this property to change count of buttons.

**Notes**

Trying to set `ButtonCount` to negative number is ignored.

**1.9.1.3.1.11 TBtnPanel.ButtonHeight Property**

Specifies the height of the buttons in array.

```
property ButtonHeight: integer;
```

**Description**

`ButtonHeight` represents the height, in pixels, of the buttons on the panel.

Setting this property to value less than 2 pixels throws an `Exception`.

By default, `ButtonHeight` is 25.

**1.9.1.3.1.12 TBtnPanel.ButtonWidth Property**

Specifies the width of the buttons in array.

```
property ButtonWidth: integer;
```

**Description**

`ButtonWidth` represents the width, in pixels, of the buttons on the panel.

Setting this property to value less than 2 pixels throws an `Exception`.

By default, `ButtonWidth` is 25.

**1.9.1.3.1.13 TBtnPanel.Color Property**

Specifies the background color of the control.

```
property Color;
```



**Description**

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

**1.9.1.3.1.14 TBtnPanel.Constraints Property**

Specifies the size constraints for the control.

```
property Constraints;
```

**Description**

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

**Warning:** Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

**1.9.1.3.1.15 TBtnPanel.Ctl3D Property**

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

```
property Ctl3D;
```

**Description**

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

**1.9.1.3.1.16 TBtnPanel.DownButton Property**

Specifies index of pressed button.

```
property DownButton: integer;
```

**Description**

Specifies index of the button which is in selected state where 0 is the first button, 1 is the second and so on;

Set this property to change selected button in the array.

By default, DownButton is -1;

**1.9.1.3.1.17 TBtnPanel.DragCursor Property**

Indicates the image used to represent the mouse pointer when the control is being dragged.

```
property DragCursor;
```

**Description**

Use the DragCursor property to change the cursor image presented when the control is being dragged.

**Note:** To make a custom cursor available for the DragCursor property, see the Cursor property.

**1.9.1.3.1.18 TBtnPanel.DragKind Property**

Specifies whether the control is being dragged normally or for docking.

```
property DragKind;
```

**Description**

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

**1.9.1.3.1.19 TBtnPanel.DragMode Property**

Determines how the control initiates drag-and-drop or drag-and-dock operations.

```
property DragMode;
```

**Description**

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

**1.9.1.3.1.20 TBtnPanel.Enabled Property**

Controls whether the control responds to mouse, keyboard, and timer events.

```
property Enabled;
```

**Description**

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

**1.9.1.3.1.21 TBtnPanel.Flat Property**

Dictates whether the button should have a 2D look instead of the usual 3D look.

```
property Flat: Boolean;
```

**Description**

Set Flat to True if you want the button to display the button without the edge bevel that gives buttons a 3D look.

By default, Flat is True.

**1.9.1.3.1.22 TBtnPanel.Font Property**

Controls the attributes of text written on or in the control.

```
property Font;
```

**Description**

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

**1.9.1.3.1.23 TBtnPanel.HintProps Property**

Provide properties to adjust hints processing.

```
property HintProps: TechHintHelper;
```

**1.9.1.3.1.24 TBtnPanel.Margins Property**

Specifies positioning of button array

```
property Margins: TBtnMargins;
```

**Description**

Margins specifies the Left, Top, Right and Bottom margins between buttons and underlying panel as well as horizontal and vertical spaces between buttons themselves.

**Example**

This example (see page 10) demonstrates how to set Margins property at run-time

**1.9.1.3.1.25 TBtnPanel.OnButtonClick Property**

Occurs when the user clicks the button on the underlying panel.

```
property OnButtonClick: TButtonClickEvent;
```

**Description**

Use the OnButtonClick event handler to respond when the user clicks the button on the TCustomBtnPanel.

OnButtonClick occurs when the user presses mouse button with the mouse pointer over the corresponding rectangle of the underlying TCustomBtnPanel.

The Sender parameter is the object whose event handler is called.

The Index parameter indicates index of pressed button.

**1.9.1.3.1.26 TBtnPanel.OnCanResize Property**

Occurs when an attempt is made to resize the control.

```
property OnCanResize;
```

**Description**

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

**1.9.1.3.1.27 TBtnPanel.OnClick Property**

Occurs when the user clicks the control.

```
property OnClick;
```

**1.9.1.3.1.28 TBtnPanel.OnConstrainedResize Property**

Adjust resize constraints.

```
property OnConstrainedResize;
```

**Description**

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

#### Notes

The OnConstrainedResize handler is called immediately after the OnCanResize handler.

### 1.9.1.3.1.29 TBtnPanel.OnContextPopup Property

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

**property** OnContextPopup;

#### Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM\_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

**Note:** Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

### 1.9.1.3.1.30 TBtnPanel.OnDbClick Property

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

**property** OnDbClick;

#### Description

Use the OnDbClick event to respond to mouse double-clicks.

### 1.9.1.3.1.31 TBtnPanel.OnDragDrop Property

Occurs when the user drops an object being dragged.

**property** OnDragDrop;

#### Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

### 1.9.1.3.1.32 TBtnPanel.OnDragOver Property

Occurs when the user drags an object over a control.

```
property OnDragOver;
```

**Description**

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

**Note:** Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

### 1.9.1.3.1.33 TBtnPanel.OnDrawButton Property

Occurs when a particular button on the panel needs to be drawn.

```
property OnDrawButton: TDrawButtonEvent;
```

**Description**

Write an OnDrawButton event handler to draw the particular button.

The Sender parameter is the object whose event handler is called.

The Rect parameter indicates the location of the button on the canvas.

The Index parameter indicates index of corresponding button button.

### 1.9.1.3.1.34 TBtnPanel.OnEndDrag Property

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

```
property OnEndDrag;
```

**Description**

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

### 1.9.1.3.1.35 TBtnPanel.OnEnter Property

Occurs when a control receives the input focus.

```
property OnEnter;
```

**Description**

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

### 1.9.1.3.1.36 TBtnPanel.OnExit Property

Occurs when the input focus shifts away from one control to another.

```
property OnExit;
```

**Description**

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

**1.9.1.3.1.37 TBtnPanel.OnGetButtonHint Property**

Occurs before hint window will be displayed.

```
property OnGetButtonHint: TGetButtonHintEvent;
```

**Description**

Use this event to specify hint text for particular button.

The Sender parameter is the object whose event handler is called.

The Index parameter indicates index of corresponding button button.

The AHint parameter used to assign Hint to be displayed for particular button.

**1.9.1.3.1.38 TBtnPanel.OnMouseDown Property**

Occurs when the user presses a mouse button with the mouse pointer over a control.

```
property OnMouseDown;
```

**Description**

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

**1.9.1.3.1.39 TBtnPanel.OnMouseMove Property**

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

```
property OnMouseMove;
```

**Description**

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

**1.9.1.3.1.40 TBtnPanel.OnMouseUp Property**

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

```
property OnMouseUp;
```

**Description**

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button

combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

#### 1.9.1.3.1.41 TBtnPanel.OnResize Property

Occurs immediately after the control is resized.

**property** OnResize;

##### Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

#### 1.9.1.3.1.42 TBtnPanel.OnStartDrag Property

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

**property** OnStartDrag;

##### Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itse

#### 1.9.1.3.1.43 TBtnPanel.Orientation Property

Specifies whether the panel's array of button is horizontal or vertical.

**property** Orientation: TRowOrientation;

##### Description

Use Orientation to specify whether the the panel's array of button is horizontal or vertical.

By default, Orientation is roHorizontal.

#### 1.9.1.3.1.44 TBtnPanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

**property** ParentBiDiMode;

#### 1.9.1.3.1.45 TBtnPanel.ParentColor Property

Determines where a control looks for its color information.

**property** ParentColor;

#### 1.9.1.3.1.46 TBtnPanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

```
property ParentCtl3D;
```

##### Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

#### 1.9.1.3.1.47 TBtnPanel.ParentFont Property

Determines where a control looks for its font information.

```
property ParentFont;
```

#### 1.9.1.3.1.48 TBtnPanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

```
property ParentShowHint;
```

#### 1.9.1.3.1.49 TBtnPanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

```
property PopupMenu;
```

##### Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

#### 1.9.1.3.1.50 TBtnPanel.RowCount Property

Indicates the number of button rows in the panel.

```
property RowCount: integer;
```

##### Description

Read RowCount to determine the number of rows in the button's array are placed.

Set RowCount to add or delete rows to rearrange buttons.

By default, RowCount is 1.

#### 1.9.1.3.1.51 TBtnPanel.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

#### 1.9.1.3.1.52 TBtnPanel.TabOrder Property

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

#### 1.9.1.3.1.53 TBtnPanel.TabStop Property

Determines if the user can tab to a control.



**property** TabStop;

**Description**

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

**1.9.1.3.1.54 TBtnPanel.Transparent Property**

Specifies whether the background of the button is transparent.

**property** Transparent: Boolean;

**Description**

Use Transparent to specify whether the background of the button is transparent.

**1.9.1.3.1.55 TBtnPanel.Visible Property**

Determines whether the component appears on screen.

**property** Visible;

**Description**


Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

**1.9.2 Structs, Records, Enums**

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

**Enumerations**

Enumeration	Description
 TRowOrientation (see page 44)	Specifies orientation of the button panel.

**Legend**

	Enumeration
-------------------------------------------------------------------------------------	-------------

**1.9.2.1 ecBtnPanel.TRowOrientation Enumeration**

```
TRowOrientation = (  
    roHorizontal,  
    roVertical,  
    roList  
);
```

**File**

ecBtnPanel

**Members**

Members	Description
roHorizontal	Horizontal from left to right, no captions
roVertical	Vertical without columns
roList	Vertical, in one column with captions

**Description**

Specifies orientation of the button panel.

### 1.9.3 Types

The following table lists types in this documentation.

**Types**

Type	Description
TButtonClickEvent (see page 45)	See TCustomBtnPanel.OnButtonClick Event (see page 27)
TDrawButtonEvent (see page 45)	See TCustomBtnPanel.OnDrawButton Event (see page 27)
TGetButtonHintEvent (see page 45)	See TCustomBtnPanel.OnGetButtonHint Event (see page 28)

#### 1.9.3.1 ecBtnPanel.TButtonClickEvent Type

```
TButtonClickEvent = procedure (Sender: TObject; Index: integer; Shift: TShiftState) of object;
```

**File**

ecBtnPanel

**Description**

See TCustomBtnPanel.OnButtonClick Event (see page 27)

#### 1.9.3.2 ecBtnPanel.TDrawButtonEvent Type

```
TDrawButtonEvent = procedure (Sender: TObject; var Rect: TRect; Index: Integer) of object;
```

**File**

ecBtnPanel

**Description**

See TCustomBtnPanel.OnDrawButton Event (see page 27)

#### 1.9.3.3 ecBtnPanel.TGetButtonHintEvent Type

```
TGetButtonHintEvent = procedure (Sender: TObject; Index: Integer; var AHint: WideString) of object;
```

**File**

ecBtnPanel

**Description**

See TCustomBtnPanel.OnGetButtonHint Event (see page 28)

## 1.10 ecDIList Namespace

## 1.10.1 Classes

The following table lists classes in this documentation.

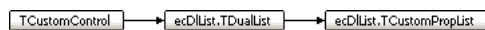
### Classes

Class	Description
TCustomPropList ( <a href="#">see page 46</a> )	Implements custom properties list.
TDualList ( <a href="#">see page 52</a> )	TDualList implements list with two columns
TPropertyItem ( <a href="#">see page 61</a> )	TPropertyItem is the base class for all nodes in the Property List (TCustomPropList ( <a href="#">see page 46</a> ))
TPropListRoot ( <a href="#">see page 65</a> )	Property items collection.

### 1.10.1.1 TCustomPropList Class

Implements custom properties list.

#### Class Hierarchy



```
TCustomPropList = class(TDualList);
```

#### File






















ecDList

#### Description


Additionally to dual list implements gutter, base property items processing, items folding.











#### Members

#### TDualList Methods















TDualList Methods	Description
 Create ( <a href="#">see page 54</a> )	Creates and initializes an instance of TDualList ( <a href="#">see page 52</a> ).
 CreateEditor ( <a href="#">see page 54</a> )	Creates in-place editor. Must be overridden in derived classes.
 CreateHandle ( <a href="#">see page 54</a> )	Creates underlying screen object.
 Destroy ( <a href="#">see page 54</a> )	Destroys an instance of TDualList ( <a href="#">see page 52</a> ).
 DoMouseWheel ( <a href="#">see page 54</a> )	Processes mouse wheel motion.
 DrawCell ( <a href="#">see page 55</a> )	Draws a specified cell.
 DrawStr ( <a href="#">see page 55</a> )	Draws string in the specified rectangle
 DrawStrW ( <a href="#">see page 55</a> )	Draws Unicode string in the specified rectangle
 ExecuteAction ( <a href="#">see page 55</a> )	Invokes an action with the component as its target.
 FocusEditor ( <a href="#">see page 56</a> )	Moves focus from list control to child in-place editor.
 IsHeaderItem ( <a href="#">see page 56</a> )	Determines if specified item is header.
 ItemRect ( <a href="#">see page 56</a> )	Returns the rectangle that surrounds the item specified in the Item parameter.
 KeyDown ( <a href="#">see page 56</a> )	Respond to key press events.
 MouseDown ( <a href="#">see page 56</a> )	Generates an OnMouseDown event.
 MouseMove ( <a href="#">see page 56</a> )	Respond to mouse moving over control area..
 MouseToItem ( <a href="#">see page 57</a> )	Returns item index depending on coordinates specified in Y parameter.
 MouseUp ( <a href="#">see page 57</a> )	Generates an OnMouseDown event.
 Paint ( <a href="#">see page 57</a> )	Renders the image of a dual list.
 SetItemIndex ( <a href="#">see page 57</a> )	Set method of ItemIndex ( <a href="#">see page 59</a> ) property.
 UpdateAction ( <a href="#">see page 57</a> )	Updates an action component to reflect the current state of the component.
 UpdateEditor ( <a href="#">see page 58</a> )	Updates current Editor ( <a href="#">see page 59</a> ).

#### TCustomPropList Class










TCustomPropList Class	Description
 Create ( <a href="#">see page 49</a> )	Creates and initializes an instance of TDualList.

  CreateItems (see page 49)	Creates root item.
 Current (see page 49)	Returns current selected item. If there is no item selected Current returns nil.
 Destroy (see page 49)	Destroys an instance of TDualList.
 DoPrepareCanvas (see page 50)	Prepares Canvas (see page 58) before painting cell.
 DrawCell (see page 50)	Draws dual list cell.
 DrawPropCell (see page 50)	Draws cell content.
 GutterWidth (see page 50)	Returns gutter width for specified row in list.
 IsHeaderItem (see page 50)	Determines if specified item is header.
 MouseDown (see page 50)	Generates an OnMouseDown event.

### TDualList Properties



TDualList Properties	Description
 BorderStyle (see page 58)	Determines the style of the line drawn around the perimeter of the panel control.
 Canvas (see page 58)	Provides access to a drawing surface that represents the TDualList (see page 52).
 Editor (see page 59)	In-place editor
 EditorVisible (see page 59)	Specifies whether editor is visible in the selected row
 ItemCount (see page 59)	Specifies the number of items in the DualList
 ItemHeight (see page 59)	Specifies the height, in pixels, of the items in the dual list
 ItemIndex (see page 59)	Specifies the index of the selected item.
 OnClick (see page 60)	Occurs when the user clicks the dual list.
 ShowGrid (see page 60)	Determines whether lines are drawn separating items in the list
 ShowSelFrame (see page 60)	Specifies whether frame rectangle should be painted around selected item.
 SplitPos (see page 60)	Specifies width of the first column in pixels (or splitter position)
 TabOrder (see page 60)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 60)	Add a summary here...
 TopItem (see page 60)	Specifies the topmost row that appears in the dual list.

### TCustomPropList Class







TCustomPropList Class	Description
 cGutter (see page 50)	Specifies color of gutter background.
 cGutterBnd (see page 51)	Specifies color of border which separates gutter from the rest of control.
 cHighlight (see page 51)	Specifies background color of selected item.
 cHighlightText (see page 51)	Specifies font color of selected item.
 FoldingIcon (see page 51)	Holds folding icon images.
 Items (see page 51)	Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.
 LeftMargin (see page 51)	Specifies left margin.
 LevelWidth (see page 51)	Specifies offset for each level of items.
 ShowGutter (see page 51)	Specifies whether gutter is visible.

### TCustomPropList Events



#### TCustomPropList Class

TCustomPropList Class	Description
 OnDrawPropCell (see page 51)	Draws cell content.
 OnGetCellParams (see page 52)	Occurs to adjust cell properties.



















### Legend

	Constructor
	virtual
	protected
	Property
	read only
	Event











**TCustomPropList Events****TCustomPropList Class**

TCustomPropList Class	Description
 OnDrawPropCell ( <a href="#">see page 51</a> )	Draws cell content.
 OnGetCellParams ( <a href="#">see page 52</a> )	Occurs to adjust cell properties.












**TDualList Methods**




TDualList Methods	Description
 Create ( <a href="#">see page 54</a> )	Creates and initializes an instance of TDualList ( <a href="#">see page 52</a> ).
 CreateEditor ( <a href="#">see page 54</a> )	Creates in-place editor. Must be overridden in derived classes.
 CreateHandle ( <a href="#">see page 54</a> )	Creates underlying screen object.
 Destroy ( <a href="#">see page 54</a> )	Destroys an instance of TDualList ( <a href="#">see page 52</a> ).
 DoMouseWheel ( <a href="#">see page 54</a> )	Processes mouse wheel motion.
 DrawCell ( <a href="#">see page 55</a> )	Draws a specified cell.
 DrawStr ( <a href="#">see page 55</a> )	Draws string in the specified rectangle
 DrawStrW ( <a href="#">see page 55</a> )	Draws Unicode string in the specified rectangle
 ExecuteAction ( <a href="#">see page 55</a> )	Invokes an action with the component as its target.
 FocusEditor ( <a href="#">see page 56</a> )	Moves focus from list control to child in-place editor.
 IsHeaderItem ( <a href="#">see page 56</a> )	Determines if specified item is header.
 ItemRect ( <a href="#">see page 56</a> )	Returns the rectangle that surrounds the item specified in the Item parameter.
 KeyDown ( <a href="#">see page 56</a> )	Respond to key press events.
 MouseDown ( <a href="#">see page 56</a> )	Generates an OnMouseUp event.
 MouseMove ( <a href="#">see page 56</a> )	Respond to mouse moving over control area..
 MouseToItem ( <a href="#">see page 57</a> )	Returns item index depending on coordinates specified in Y parameter.
 MouseUp ( <a href="#">see page 57</a> )	Generates an OnMouseUp event.
 Paint ( <a href="#">see page 57</a> )	Renders the image of a dual list.
 SetItemIndex ( <a href="#">see page 57</a> )	Set method of ItemIndex ( <a href="#">see page 59</a> ) property.
 UpdateAction ( <a href="#">see page 57</a> )	Updates an action component to reflect the current state of the component.
 UpdateEditor ( <a href="#">see page 58</a> )	Updates current Editor ( <a href="#">see page 59</a> ).

**TCustomPropList Class**










TCustomPropList Class	Description
 Create ( <a href="#">see page 49</a> )	Creates and initializes an instance of TDualList.
 CreateItems ( <a href="#">see page 49</a> )	Creates root item.
 Current ( <a href="#">see page 49</a> )	Returns current selected item. If there is no item selected Current returns nil.
 Destroy ( <a href="#">see page 49</a> )	Destroys an instance of TDualList.
 DoPrepareCanvas ( <a href="#">see page 50</a> )	Prepares Canvas ( <a href="#">see page 58</a> ) before painting cell.
 DrawCell ( <a href="#">see page 50</a> )	Draws dual list cell.
 DrawPropCell ( <a href="#">see page 50</a> )	Draws cell content.
 GutterWidth ( <a href="#">see page 50</a> )	Returns gutter width for specified row in list.
 IsHeaderItem ( <a href="#">see page 50</a> )	Determines if specified item is header.
 MouseDown ( <a href="#">see page 50</a> )	Generates an OnMouseUp event.

**TDualList Properties**

TDualList Properties	Description
 BorderStyle ( <a href="#">see page 58</a> )	Determines the style of the line drawn around the perimeter of the panel control.
 Canvas ( <a href="#">see page 58</a> )	Provides access to a drawing surface that represents the TDualList ( <a href="#">see page 52</a> ).
 Editor ( <a href="#">see page 59</a> )	In-place editor
 EditorVisible ( <a href="#">see page 59</a> )	Specifies whether editor is visible in the selected row
 ItemCount ( <a href="#">see page 59</a> )	Specifies the number of items in the DualList
 ItemHeight ( <a href="#">see page 59</a> )	Specifies the height, in pixels, of the items in the dual list
 ItemIndex ( <a href="#">see page 59</a> )	Specifies the index of the selected item.
 OnClick ( <a href="#">see page 60</a> )	Occurs when the user clicks the dual list.
 ShowGrid ( <a href="#">see page 60</a> )	Determines whether lines are drawn separating items in the list
 ShowSelFrame ( <a href="#">see page 60</a> )	Specifies whether frame rectangle should be painted around selected item.
 SplitPos ( <a href="#">see page 60</a> )	Specifies width of the first column in pixels (or splitter position)

 TabOrder (see page 60)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 60)	Add a summary here...
 TopItem (see page 60)	Specifies the topmost row that appears in the dual list.

### TCustomPropList Class

TCustomPropList Class	Description
 cGutter (see page 50)	Specifies color of gutter background.
 cGutterBnd (see page 51)	Specifies color of border which separates gutter from the rest of control.
 cHighlight (see page 51)	Specifies background color of selected item.
 cHighlightText (see page 51)	Specifies font color of selected item.
 FoldingIcon (see page 51)	Holds folding icon images.
 Items (see page 51)	Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.
 LeftMargin (see page 51)	Specifies left margin.
 LevelWidth (see page 51)	Specifies offset for each level of items.
 ShowGutter (see page 51)	Specifies whether gutter is visible.

## 1.10.1.1.1 TCustomPropList Methods

### 1.10.1.1.1.1 TCustomPropList.Create Constructor

Creates and initializes an instance of TDualList.

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

#### Description

Calling Create constructs and initializes an instance of TDualList.

After calling the inherited TCustomControl constructor, Create initializes the dual list.

### 1.10.1.1.1.2 TCustomPropList.CreateItems Method

Creates root item.

```
function CreateItems: TPropListRoot; virtual;
```

#### Description

CreateItems creates items storage.

### 1.10.1.1.1.3 TCustomPropList.Current Method

Returns current selected item. If there is no item selected Current returns nil.

```
function Current: TPropertyItem;
```

### 1.10.1.1.1.4 TCustomPropList.Destroy Destructor

Destroys an instance of TDualList.

```
destructor Destroy; override;
```

#### Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the dual list is destroyed, it destroys all the objects it owns and then call inherited Destroy procedure;

### 1.10.1.1.1.5 TCustomPropList.DoPrepareCanvas Method

Prepares Canvas (see page 58) before painting cell.

```
procedure DoPrepareCanvas(Node: TPropertyItem; CellType: TCellType; var Alignment: TAlignment); virtual;
```

#### Description

Node and CellType specifies position in list. Change property of control Canvas (see page 58) and Alignment parameter to change cell painting style.

### 1.10.1.1.1.6 TCustomPropList.DrawCell Method

Draws dual list cell.

```
procedure DrawCell(Rect: TRect; ACol: Integer; ARow: Integer; Selected: Boolean); override;
```

### 1.10.1.1.1.7 TCustomPropList.DrawPropCell Method

Draws cell content.

```
procedure DrawPropCell(const R: TRect; Node: TPropertyItem; CellType: TCellType; Alignment: TAlignment); virtual;
```

### 1.10.1.1.1.8 TCustomPropList.GutterWidth Method

Returns gutter width for specified row in list.

```
function GutterWidth(Row: integer): integer;
```

### 1.10.1.1.1.9 TCustomPropList.IsHeaderItem Method

Determines if specified item is header.

```
function IsHeaderItem(Index: integer): Boolean; override;
```

#### Description

If IsHeaderItem is True, item is header, i.e. it occupies two columns and in-place editor is not created for this one.

In TDualList it always returns false. Descendants must override this function to make result sensible.

### 1.10.1.1.1.10 TCustomPropList.MouseDown Method

Generates an OnMouseUp event.

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

#### Description

Overridden from TControl.MouseDown.

After calls inherited procedure dual list determines item at mouse and adjusts ItemIndex property

## 1.10.1.1.2 TCustomPropList Properties

### 1.10.1.1.2.1 TCustomPropList.cGutter Property

Specifies color of gutter background.

```
property cGutter: TColor;
```

#### 1.10.1.1.2.2 TCustomPropList.cGutterBnd Property

Specifies color of border which separates gutter from the rest of control.

```
property cGutterBnd: TColor;
```

#### 1.10.1.1.2.3 TCustomPropList.cHighlight Property

Specifies background color of selected item.

```
property cHighlight: TColor;
```

#### 1.10.1.1.2.4 TCustomPropList.cHighlightText Property

Specifies font color of selected item.

```
property cHighlightText: TColor;
```

#### 1.10.1.1.2.5 TCustomPropList.FoldingIcon Property

Holds folding icon images.

```
property FoldingIcon: TBitmap;
```

##### Description

FoldingIcon should contain two images in a row, first - collapse icon (-), second - expand icon (+).

Color of bottom-left pixel is used as mask color.

Folding icon is initialized from resource when control is created at design time.

#### 1.10.1.1.2.6 TCustomPropList.Items Property

Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.

```
property Items: TPropListRoot;
```

#### 1.10.1.1.2.7 TCustomPropList.LeftMargin Property

Specifies left margin.

```
property LeftMargin: integer;
```

#### 1.10.1.1.2.8 TCustomPropList.LevelWidth Property

Specifies offset for each level of items.

```
property LevelWidth: integer;
```

#### 1.10.1.1.2.9 TCustomPropList.ShowGutter Property

Specifies whether gutter is visible.

```
property ShowGutter: Boolean;
```

### 1.10.1.1.3 TCustomPropList Events

#### 1.10.1.1.3.1 TCustomPropList.OnDrawPropCell Event

Draws cell content.

```
property OnDrawPropCell: TCustomPropDrawEvent;
```

##### Description

Write OnDrawPropCell event handler to implement custom drawing of the cell content. If OnDrawPropCell event is assigned control does not perform default drawing, but it prepares Canvas (see page 58) for drawing.



### 1.10.1.1.3.2 TCustomPropList.OnGetCellParams Event

Occurs to adjust cell properties.

**property** OnGetCellParams: TGetCellParamsEvent;

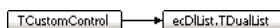
#### Description

Write OnGetCellParams event handler to change cell properties by assigning new values to Alignment parameter and to Pen, Font and Brush of the property list Canvas (see page 58).

## 1.10.1.2 TDualList Class

TDualList implements list with two columns

#### Class Hierarchy



```
TDualList = class(TCustomControl);
```

#### File

ecDList

#### Description

TDualList is used as the base class for TCustomInspectorList. It implements list with two columns. First column is used for names, second one for values. In-place editor are used to edit values in the second column.











#### Members

#### TDualList Methods






TDualList Methods	Description
Create (see page 54)	Creates and initializes an instance of TDualList.
CreateEditor (see page 54)	Creates in-place editor. Must be overridden in derived classes.
CreateHandle (see page 54)	Creates underlying screen object.
Destroy (see page 54)	Destroys an instance of TDualList.
DoMouseWheel (see page 54)	Processes mouse wheel motion.
DrawCell (see page 55)	Draws a specified cell.
DrawStr (see page 55)	Draws string in the specified rectangle
DrawStrW (see page 55)	Draws Unicode string in the specified rectangle
ExecuteAction (see page 55)	Invokes an action with the component as its target.
FocusEditor (see page 56)	Moves focus from list control to child in-place editor.
IsHeaderItem (see page 56)	Determines if specified item is header.
ItemRect (see page 56)	Returns the rectangle that surrounds the item specified in the Item parameter.
KeyDown (see page 56)	Respond to key press events.
MouseDown (see page 56)	Generates an OnMouseUp event.
MouseMove (see page 56)	Respond to mouse moving over control area..
MouseToItem (see page 57)	Returns item index depending on coordinates specified in Y parameter.
MouseUp (see page 57)	Generates an OnMouseUp event.
Paint (see page 57)	Renders the image of a dual list.
SetItemIndex (see page 57)	Set method of ItemIndex (see page 59) property.
UpdateAction (see page 57)	Updates an action component to reflect the current state of the component.
UpdateEditor (see page 58)	Updates current Editor (see page 59).

#### TDualList Properties






















TDualList Properties	Description
BorderStyle (see page 58)	Determines the style of the line drawn around the perimeter of the panel control.
Canvas (see page 58)	Provides access to a drawing surface that represents the TDualList.
Editor (see page 59)	In-place editor
EditorVisible (see page 59)	Specifies whether editor is visible in the selected row

 ItemCount (see page 59)	Specifies the number of items in the DualList
 ItemHeight (see page 59)	Specifies the height, in pixels, of the items in the dual list
 ItemIndex (see page 59)	Specifies the index of the selected item.
 OnClick (see page 60)	Occurs when the user clicks the dual list.
 ShowGrid (see page 60)	Determines whether lines are drawn separating items in the list
 ShowSelFrame (see page 60)	Specifies whether frame rectangle should be painted around selected item.
 SplitPos (see page 60)	Specifies width of the first column in pixels (or splitter position)
 TabOrder (see page 60)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 60)	Add a summary here...
 TopItem (see page 60)	Specifies the topmost row that appears in the dual list.















## Legend

	Constructor
	virtual
	protected
	Property
	read only

## TDualList Methods

TDualList Methods	Description
 Create (see page 54)	Creates and initializes an instance of TDualList.
 CreateEditor (see page 54)	Creates in-place editor. Must be overridden in derived classes.
 CreateHandle (see page 54)	Creates underlying screen object.
 Destroy (see page 54)	Destroys an instance of TDualList.
 DoMouseWheel (see page 54)	Processes mouse wheel motion.
 DrawCell (see page 55)	Draws a specified cell.
 DrawStr (see page 55)	Draws string in the specified rectangle
 DrawStrW (see page 55)	Draws Unicode string in the specified rectangle
 ExecuteAction (see page 55)	Invokes an action with the component as its target.
 FocusEditor (see page 56)	Moves focus from list control to child in-place editor.
 IsHeaderItem (see page 56)	Determines if specified item is header.
 ItemRect (see page 56)	Returns the rectangle that surrounds the item specified in the Item parameter.
 KeyDown (see page 56)	Respond to key press events.
 MouseDown (see page 56)	Generates an OnMouseDown event.
 MouseMove (see page 56)	Respond to mouse moving over control area..
 MouseToItem (see page 57)	Returns item index depending on coordinates specified in Y parameter.
 MouseUp (see page 57)	Generates an OnMouseUp event.
 Paint (see page 57)	Renders the image of a dual list.
 SetItemIndex (see page 57)	Set method of ItemIndex (see page 59) property.
 UpdateAction (see page 57)	Updates an action component to reflect the current state of the component.
 UpdateEditor (see page 58)	Updates current Editor (see page 59).

## TDualList Properties

TDualList Properties	Description
 BorderStyle (see page 58)	Determines the style of the line drawn around the perimeter of the panel control.
 Canvas (see page 58)	Provides access to a drawing surface that represents the TDualList.
 Editor (see page 59)	In-place editor
 EditorVisible (see page 59)	Specifies whether editor is visible in the selected row
 ItemCount (see page 59)	Specifies the number of items in the DualList
 ItemHeight (see page 59)	Specifies the height, in pixels, of the items in the dual list
 ItemIndex (see page 59)	Specifies the index of the selected item.
 OnClick (see page 60)	Occurs when the user clicks the dual list.
 ShowGrid (see page 60)	Determines whether lines are drawn separating items in the list
 ShowSelFrame (see page 60)	Specifies whether frame rectangle should be painted around selected item.
 SplitPos (see page 60)	Specifies width of the first column in pixels (or splitter position)
 TabOrder (see page 60)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 60)	Add a summary here...
 TopItem (see page 60)	Specifies the topmost row that appears in the dual list.

## 1.10.1.2.1 TDualList Methods

### 1.10.1.2.1.1 TDualList.Create Constructor

Creates and initializes an instance of TDualList (see page 52).

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

#### Description

Calling Create constructs and initializes an instance of TDualList (see page 52).

After calling the inherited TCustomControl constructor, Create initializes the dual list.

### 1.10.1.2.1.2 TDualList.CreateEditor Method

Creates in-place editor. Must be overridden in derived classes.

```
function CreateEditor: TCustomEditEx; virtual;
```

#### Description

This procedure creates in-place editor. It must be overridden in derived classes.

### 1.10.1.2.1.3 TDualList.CreateHandle Method

Creates underlying screen object.

```
procedure CreateHandle; override;
```

#### Description

This is overridden procedure from TWinControl.CreateHandle.

After calling inherited TWinControl.CreateHandle it adjusts dual list representation so that current Editor (see page 59) will be visible and focused.

### 1.10.1.2.1.4 TDualList.Destroy Destructor

Destroys an instance of TDualList (see page 52).

```
destructor Destroy; override;
```

#### Description

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

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

As the dual list is destroyed, it destroys all the objects it owns and then call inherited Destroy procedure;

### 1.10.1.2.1.5 TDualList.DoMouseWheel Method

Processes mouse wheel motion.

```
function DoMouseWheel(Shift: TShiftState; WheelDelta: Integer; MousePos: TPoint): Boolean;  
override;
```

#### Description

Overridden from TControl.DoMouseWheel.

First dual list calls inherited TControl.DoMouseWheel procedure and then adjust TopItem (see page 60) property;

#### 1.10.1.2.1.6 TDualList.DrawCell Method

Draws a specified cell.

```
procedure DrawCell(Rect: TRect; ACol: integer; ARow: integer; Selected: Boolean); virtual;
```

##### Description

In the TDualList (see page 52) it does nothing. Descendants must override this method to do some useful things.

#### 1.10.1.2.1.7 TDualList.DrawStr Method

Draws string in the specified rectangle

```
procedure DrawStr(Rect: TRect; const S: string; Align: TAlignment);
```

##### Description

Draws string in the specified rectangle. Rect specifies the rectangle for output, S specifies text string, and Align specifies how text is aligned when draws.

This procedure internally calls Win32API method DrawText.

#### 1.10.1.2.1.8 TDualList.DrawStrW Method

Draws Unicode string in the specified rectangle

```
procedure DrawStrW(Rect: TRect; const S: WideString; Align: TAlignment);
```

##### Description

Draws Unicode string in the specified rectangle. Rect specifies the rectangle for output, S specifies text string, and Align specifies how text is aligned when draws.

This procedure internally calls Win32API method DrawTextW.

#### 1.10.1.2.1.9 TDualList.ExecuteAction Method

Invokes an action with the component as its target.

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

##### Description

When the user invokes an action, VCL makes a series of calls to respond to that action. First, it generates an OnExecute event of the action list that contains the action. If the action list does not handle the OnExecute event, then the action is routed to the Application object's ExecuteAction method, which invokes the OnActionExecute event handler. If the OnActionExecute event handler does not handle the action, then it is routed to the action's OnExecute event handler. If that does not handle the action, the active control's ExecuteAction method is called.

The Action parameter specifies the action that was invoked. ExecuteAction returns true if the action was successfully dispatched, and false if the component could not handle the action. If ExecuteAction returns false for the active control, VCL calls the active form's ExecuteAction method. If this returns false, VCL tries all active controls in the form. If these all return false, VCL repeats the process with the main form, if that is different from the active form.

#### 1.10.1.2.1.10 TDualList.FocusEditor Method

Moves focus from list control to child in-place editor.

```
procedure FocusEditor; virtual;
```

#### 1.10.1.2.1.11 TDualList.IsHeaderItem Method

Determines if specified item is header.

```
function IsHeaderItem(Index: integer): Boolean; virtual;
```

##### Description

If IsHeaderItem is True, item is header, i.e. it occupies two columns and in-place editor is not created for this one.

In TDualList (see page 52) it always returns false. Descendants must override this function to make result sensible.

#### 1.10.1.2.1.12 TDualList.ItemRect Method

Returns the rectangle that surrounds the item specified in the Item parameter.

```
function ItemRect(Index: integer): TRect;
```

##### Description

Use ItemRect to get the coordinates of a particular item in the dual list.

The Item parameter is the ItemIndex (see page 59) of the item whose position is queried.

#### 1.10.1.2.1.13 TDualList.KeyDown Method

Respond to key press events.

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

##### Description

Overridden from TWinControl.KeyDown.

After call the inherited method it determines Key parameter and adjust ItemIndex (see page 59) and TopItem (see page 60) property.

#### 1.10.1.2.1.14 TDualList.MouseDown Method

Generates an OnMouseUp event.

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
override;
```

##### Description

Overridden from TControl.MouseDown.

After calls inherited procedure dual list determines item at mouse and adjusts ItemIndex (see page 59) property

#### 1.10.1.2.1.15 TDualList.MouseMove Method

Respond to mouse moving over control area..

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

**Description**

Overridden from TControl.MouseDown (see page 56).

After calling inherited procedure dual list do next:

- if "virtual splitter" is captured then adjusts SplitPos (see page 60)
- else determines item at mouse and adjusts ItemIndex (see page 59);

**1.10.1.2.1.16 TDualList.MouseToItem Method**

Returns item index depending on coordinates specified in Y parameter.

```
function MouseToItem(Y: integer): integer;
```

**Description**

Returns item index for the given Y coordinate. If Y coordinate is out of items, MouseToItem returns -1

**1.10.1.2.1.17 TDualList.MouseUp Method**

Generates an OnMouseUp event.

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
override;
```

**Description**

Overridden from TControl.MouseUp.

After calling inherited procedure dual list releases mouse capture, "virtual splitter" and stops the inner timer.

**1.10.1.2.1.18 TDualList.Paint Method**

Renders the image of a dual list.

```
procedure Paint; override;
```

**Description**

Overridden from TCustomControl.Paint

After calling inherited procedure dual list do all its specified painting tasks.

**1.10.1.2.1.19 TDualList.SetItemIndex Method**

Set method of ItemIndex (see page 59) property.

```
procedure SetItemIndex(Value: integer); virtual;
```

**Description**

Derived classes may change default behavior of changing item, for example, to restrict some indexes.

**1.10.1.2.1.20 TDualList.UpdateAction Method**

Updates an action component to reflect the current state of the component.

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

**Description**

When the application is idle, VCL makes a series of calls to update the properties (such as whether it is enabled, checked,

and so on) of every action that is linked to a visible control or menu item. First, VCL generates an OnUpdate event of the action list that contains the action. If the action list does not handle the OnUpdate event, then the action is routed to the Application object's UpdateAction method, which invokes the OnActionUpdate event handler. If the OnActionUpdate event handler does not update the action, then it is routed to the action's OnUpdate event handler. If that does not update the action, the active control's UpdateAction method is called.

The Action parameter specifies the action component that should be updated. UpdateAction returns true if the action component now reflects the state of the component, and false if it did not know how to update the action. If UpdateAction returns false for the active component, VCL calls the active form's

s UpdateAction method.

Do not call UpdateAction. It is called automatically when the application is idle. As implemented in TComponent, UpdateAction allows the action to update itself with the component as a target. Descendants can override this method to perform updates that reflect class-specific properties or states.

#### 1.10.1.2.1.21 TDualList.UpdateEditor Method

Updates current Editor (see page 59).

```
procedure UpdateEditor; virtual;
```

##### Description

This procedure clears Text and EditMask properties and sets MaxLength to 0.

Descendants must override this to make it more sensible.

#### 1.10.1.2.2 TDualList Properties

##### 1.10.1.2.2.1 TDualList.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

```
property BorderStyle: TBorderStyle;
```

##### Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

##### 1.10.1.2.2.2 TDualList.Canvas Property

Provides access to a drawing surface that represents the TDualList (see page 52).

```
property Canvas;
```

##### Description

This property is inherited and redeclared as public for using in descendants of TDualList (see page 52).

### 1.10.1.2.2.3 TDualList.Editor Property

In-place editor

```
property Editor: TCustomEditEx;
```

#### Description

Represents in-place editor for selected row in list. Depending on EditStyle it can be simple, with ellipsis (...) button or with drop-down list.

### 1.10.1.2.2.4 TDualList.EditorVisible Property

Specifies whether editor is visible in the selected row

```
property EditorVisible: Boolean;
```

#### Description

Specifies whether editor is visible or not in the selected row.

This property may become False when corresponding property node is not a TPropertyNode type. For example when user switches object inspector's "Property" tab in "By Category" mode.

### 1.10.1.2.2.5 TDualList.ItemCount Property

Specifies the number of items in the DualList

```
property ItemCount: integer;
```

#### Description

Specifies the number of items in the DualList.

Read ItemCount to get the number of items in the dual list. ItemCount is the number of all items, not just those that are visible at one time.

### 1.10.1.2.2.6 TDualList.ItemHeight Property

Specifies the height, in pixels, of the items in the dual list

```
property ItemHeight: integer;
```

#### Description

Read ItemHeight to determine the height of the items in the dual list. Set this property to change height of the items.

### 1.10.1.2.2.7 TDualList.ItemIndex Property

Specifies the index of the selected item.

```
property ItemIndex: integer;
```

#### Description

Read ItemIndex to determine which item is selected. The first item in the list has index 0, the second item has index 1, and so on.

Set ItemIndex programmatically to select an item by passing in the index value.

If new value less than 0 and ItemCount (see page 59) > 0 then ItemIndex will be 0,

else if new value exceeds ItemCount (see page 59)-1 then ItemIndex will be ItemCount (see page 59)-1.

Else ItemIndex will be set to new value.



#### 1.10.1.2.2.8 TDualList.OnClick Property

Occurs when the user clicks the dual list.

```
property OnClick;
```

##### Description

OnClick is inherited event from TControl.

Use the OnClick event handler to respond when the user clicks the dual list.

#### 1.10.1.2.2.9 TDualList.ShowGrid Property

Determines whether lines are drawn separating items in the list

```
property ShowGrid: Boolean;
```

##### Description

Specifies whether horizontal grid lines is visible.

Set ShowGrid to True to add lines that separate the items in the dual list.

#### 1.10.1.2.2.10 TDualList.ShowSelFrame Property

Specifies whether frame rectangle should be painted around selected item.

```
property ShowSelFrame: Boolean;
```

#### 1.10.1.2.2.11 TDualList.SplitPos Property

Specifies width of the first column in pixels (or splitter position)

```
property SplitPos: integer;
```

##### Description

Specifies width of the first column (or splitter position). Width of the second column equals to Width - SplitPos.

Set SplitPos programmatically to change width of columns.

Minimal value of SplitPos is 20;

#### 1.10.1.2.2.12 TDualList.TabOrder Property

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

##### Description

It is inherited property from TWinControl. See the TWinControl.TabOrder for description.

#### 1.10.1.2.2.13 TDualList.TabStop Property

Add a summary here...

```
property TabStop;
```

##### Description

It is inherited property from TWinControl. See the TWinControl.Stop for description.

#### 1.10.1.2.2.14 TDualList.TopItem Property

Specifies the topmost row that appears in the dual list.

```
property TopItem: integer;
```

### Description

When TopItem is changed, the dual list scrolls vertically so that the specified row is topmost in the view.

## 1.10.1.3 TPropertyItem Class

TPropertyItem is the base class for all nodes in the Property List (TCustomPropList (see page 46))

### Class Hierarchy

```
ecDIList.TPropertyItem
```

```
TPropertyItem = class;
```

### File
























ecDIList

### Description










TPropNodeBase introduces base functionality.

### Members





#### TPropertyItem Methods

TPropertyItem Methods	Description
 Add (see page 62)	Adds new property item.
  Changed (see page 62)	Called when property item was changed.
  Clear (see page 62)	Deletes all items from the node.
 Create (see page 62)	Creates and initializes a TPropertyItem instance.
 Delete (see page 63)	Deletes Item at index
  Destroy (see page 63)	Destroys an instance of TPropertyItem.
  Expandable (see page 63)	Specifies whether property item can be expanded.
  GetName (see page 63)	Returns name of the item. May be overridden in derived class.
  HasValue (see page 63)	Specifies whether property item has value. For example, category item does not have value.
 IndexOf (see page 63)	Returns index of child item. If Item is not a child returns -1.
 Insert (see page 63)	Adds a property item to the Items (see page 64) array at the position specified by Index.
  IsEqual (see page 63)	Returns True if property items are equal.
  IsRoot (see page 63)	Returns True if the item is root (see page 64) item.
 Move (see page 63)	Changes the position of an item in the Items (see page 64) array.
 Root (see page 64)	Specifies Root item.

#### TPropertyItem Properties
















TPropertyItem Properties	Description
 Count (see page 64)	Determines count of child items.
 DisplayName (see page 64)	Specifies name displayed on screen
 Expanded (see page 64)	Specifies if node is expanded or not.
 Items (see page 64)	Provides indexed access to the child items.
 Level (see page 64)	Indicates the level of indentation of a item within the property list control..
 Name (see page 64)	Specifies the name of the property node.
 Parent (see page 65)	Indicates the parent property of the node.
 PathName (see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
 Visible (see page 65)	Specifies whether item is selected.

### Legend










	Method
	protected
	virtual
	Property

	read only
-----------------------------------------------------------------------------------	-----------

## TPropertyItem Methods

TPropertyItem Methods	Description
 Add ( <a href="#">see page 62</a> )	Adds new property item.
 Changed ( <a href="#">see page 62</a> )	Called when property item was changed.
 Clear ( <a href="#">see page 62</a> )	Deletes all items from the node.
 Create ( <a href="#">see page 62</a> )	Creates and initializes a TPropertyItem instance.
 Delete ( <a href="#">see page 63</a> )	Deletes Item at index
 Destroy ( <a href="#">see page 63</a> )	Destroys an instance of TPropertyItem.
 Expandable ( <a href="#">see page 63</a> )	Specifies whether property item can be expanded.
 GetName ( <a href="#">see page 63</a> )	Returns name of the item. May be overridden in derived class.
 HasValue ( <a href="#">see page 63</a> )	Specifies whether property item has value. For example, category item does not have value.
 IndexOf ( <a href="#">see page 63</a> )	Returns index of child item. If Item is not a child returns -1.
 Insert ( <a href="#">see page 63</a> )	Adds a property item to the Items ( <a href="#">see page 64</a> ) array at the position specified by Index.
 IsEqual ( <a href="#">see page 63</a> )	Returns True if property items are equal.
 IsRoot ( <a href="#">see page 63</a> )	Returns True if the item is root ( <a href="#">see page 64</a> ) item.
 Move ( <a href="#">see page 63</a> )	Changes the position of an item in the Items ( <a href="#">see page 64</a> ) array.
 Root ( <a href="#">see page 64</a> )	Specifies Root item.

## TPropertyItem Properties

TPropertyItem Properties	Description
 Count ( <a href="#">see page 64</a> )	Determines count of child items.
 DisplayName ( <a href="#">see page 64</a> )	Specifies name displayed on screen
 Expanded ( <a href="#">see page 64</a> )	Specifies if node is expanded or not.
 Items ( <a href="#">see page 64</a> )	Provides indexed access to the child items.
 Level ( <a href="#">see page 64</a> )	Indicates the level of indentation of a item within the property list control..
 Name ( <a href="#">see page 64</a> )	Specifies the name of the property node.
 Parent ( <a href="#">see page 65</a> )	Indicates the parent property of the node.
 PathName ( <a href="#">see page 65</a> )	Returns path of the item. Path is combined from the item name and all parent names.
 Visible ( <a href="#">see page 65</a> )	Specifies whether item is selected.

### 1.10.1.3.1 TPropertyItem Methods

#### 1.10.1.3.1.1 TPropertyItem.Add Method

Adds new property item.

```
procedure Add(Item: TPropertyItem);
```

#### 1.10.1.3.1.2 TPropertyItem.Changed Method

Called when property item was changed.

```
procedure Changed; virtual;
```

#### 1.10.1.3.1.3 TPropertyItem.Clear Method

Deletes all items from the node.

```
procedure Clear; virtual;
```

#### 1.10.1.3.1.4 TPropertyItem.Create Constructor

Creates and initializes a TPropertyItem instance.

```
constructor Create;
```

**Description**

Use Create to programmatically instantiate a TPropertyItem object.

**1.10.1.3.1.5 TPropertyItem.Delete Method**

Deletes Item at index

```
procedure Delete(Index: integer);
```

**1.10.1.3.1.6 TPropertyItem.Destroy Destructor**

Destroys an instance of TPropertyItem.

```
destructor Destroy; override;
```

**Description**

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

**1.10.1.3.1.7 TPropertyItem.Expandable Method**

Specifies whether property item can be expanded.

```
function Expandable: Boolean; virtual;
```

**1.10.1.3.1.8 TPropertyItem.GetName Method**

Returns name of the item. May be overridden in derived class.

```
function GetName: string; virtual;
```

**1.10.1.3.1.9 TPropertyItem.HasValue Method**

Specifies whether property item has value. For example, category item does not have value.

```
function HasValue: Boolean; virtual;
```

**1.10.1.3.1.10 TPropertyItem.IndexOf Method**

Returns index of child item. If Item is not a child returns -1.

```
function IndexOf(Item: TPropertyItem): integer;
```

**1.10.1.3.1.11 TPropertyItem.Insert Method**

Adds a property item to the Items (☐ see page 64) array at the position specified by Index.

```
procedure Insert(Index: integer; Item: TPropertyItem);
```

**1.10.1.3.1.12 TPropertyItem.IsEqual Method**

Returns True if property items are equal.

```
function IsEqual(Other: TPropertyItem): Boolean; virtual;
```

**1.10.1.3.1.13 TPropertyItem.IsRoot Method**

Returns True if the item is root (☐ see page 64) item.

```
function IsRoot: Boolean;
```

**1.10.1.3.1.14 TPropertyItem.Move Method**

Changes the position of an item in the Items (☐ see page 64) array.

```
procedure Move(CurIndex: integer; NewIndex: integer);
```

**Description**

Call Move to move the item at the position CurlIndex so that it occupies the position NewIndex. CurlIndex and NewIndex are zero-based indexes into the Items (see page 64) array.

**1.10.1.3.1.15 TPropertyItem.Root Method**

Specifies Root item.

```
function Root: TPropertyItem;
```

**1.10.1.3.2 TPropertyItem Properties****1.10.1.3.2.1 TPropertyItem.Count Property**

Determines count of child items.

```
property Count: integer;
```

**Description**

This property have meaning only when node is expanded.

**1.10.1.3.2.2 TPropertyItem.DisplayName Property**

Specifies name displayed on screen

```
property DisplayName: WideString;
```

**Description**

This name may differ from Name (see page 64) property (due to localization e.g.)

**1.10.1.3.2.3 TPropertyItem.Expanded Property**

Specifies if node is expanded or not.

```
property Expanded: Boolean;
```

**Description**

Set this property to True to make the node expanded or to False to close it up.

**1.10.1.3.2.4 TPropertyItem.Items Property**

Provides indexed access to the child items.

```
property Items [Index: integer]: TPropertyItem;
```

**Description**

Use Item to access a child node based on its Index property. The first child node has an index of 0, the second an index of 1, and so on.

**1.10.1.3.2.5 TPropertyItem.Level Property**

Indicates the level of indentation of a item within the property list control..

```
property Level: integer;
```

**1.10.1.3.2.6 TPropertyItem.Name Property**

Specifies the name of the property node.

```
property Name: string;
```

**Description**

In this class Name returns an empty string.

This property overrides in descendants to become sensible.

**1.10.1.3.2.7 TPropertyItem.Parent Property**

Indicates the parent property of the node.

```
property Parent: TPropertyItem;
```

**Description**

Use the Parent property to get the parent node of this control.

**1.10.1.3.2.8 TPropertyItem.PathName Property**

Returns path of the item. Path is combined from the item name and all parent names.

```
property PathName: string;
```

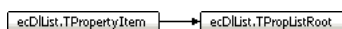
**1.10.1.3.2.9 TPropertyItem.Visible Property**

Specifies whether item is selected.

```
property Visible: Boolean;
```

**1.10.1.4 TPropListRoot Class**

Property items collection.

**Class Hierarchy**

```
TPropListRoot = class(TPropertyItem);
```

**File**


ecDIList

**Description**











Provides access to property items; manages expanding of items and items updating.

**Members****TPropertyItem Methods**










TPropertyItem Methods	Description
➤ Add (🔗 see page 62)	Adds new property item.
➤ 🔗 Changed (🔗 see page 62)	Called when property item was changed.
➤ 🔗 Clear (🔗 see page 62)	Deletes all items from the node.
➤ 🔗 Create (🔗 see page 62)	Creates and initializes a TPropertyItem instance.
➤ 🔗 Delete (🔗 see page 63)	Deletes Item at index
➤ 🔗 Destroy (🔗 see page 63)	Destroys an instance of TPropertyItem.
➤ 🔗 Expandable (🔗 see page 63)	Specifies whether property item can be expanded.
➤ 🔗 GetName (🔗 see page 63)	Returns name of the item. May be overridden in derived class.
➤ 🔗 HasValue (🔗 see page 63)	Specifies whether property item has value. For example, category item does not have value.
➤ 🔗 IndexOf (🔗 see page 63)	Returns index of child item. If Item is not a child returns -1.
➤ 🔗 Insert (🔗 see page 63)	Adds a property item to the Items (🔗 see page 64) array at the position specified by Index.
➤ 🔗 IsEqual (🔗 see page 63)	Returns True if property items are equal.
➤ 🔗 IsRoot (🔗 see page 63)	Returns True if the item is root (🔗 see page 64) item.
➤ 🔗 Move (🔗 see page 63)	Changes the position of an item in the Items (🔗 see page 64) array.

 Root (see page 64)	Specifies Root item.
------------------------------------------------------------------------------------------------------	----------------------




**TPropListRoot Class**

TPropListRoot Class	Description
 BeginUpdate (see page 67)	Suspends updating of property list.
 Changed (see page 67)	Called when property item was changed.
 Create (see page 67)	Creates and initializes a TPropListRoot instance.
 Destroy (see page 68)	Destroys an instance of TPropListRoot.
 EndUpdate (see page 68)	Re-enables screen repainting.
 ExpandItem (see page 68)	Called when item is to be expanded.
 ExpIndexOf (see page 68)	Returns visible index (expanded) of specified property node.
 RestoreState (see page 68)	Restores previously saved state.
 SaveState (see page 68)	Saves state, i.e. expanded items and select item.
 UpdateList (see page 68)	Updates property list.






**TPropertyItem Properties**

TPropertyItem Properties	Description
 Count (see page 64)	Determines count of child items.
 DisplayName (see page 64)	Specifies name displayed on screen
 Expanded (see page 64)	Specifies if node is expanded or not.
 Items (see page 64)	Provides indexed access to the child items.
 Level (see page 64)	Indicates the level of indentation of a item within the property list control..
 Name (see page 64)	Specifies the name of the property node.
 Parent (see page 65)	Indicates the parent property of the node.
 PathName (see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
 Visible (see page 65)	Specifies whether item is selected.
















**TPropListRoot Class**

TPropListRoot Class	Description
 ExpCount (see page 68)	Returns count of child nodes and their nodes recursively
 ExplItems (see page 68)	Provides indexed access to the list of visible items.
 Owner (see page 69)	Specifies custom property list - owner of property items collection.











**Legend**

	Method
	protected
	virtual
	Property
	read only










**TPropertyItem Methods**

TPropertyItem Methods	Description
 Add (see page 62)	Adds new property item.
 Changed (see page 62)	Called when property item was changed.
 Clear (see page 62)	Deletes all items from the node.
 Create (see page 62)	Creates and initializes a TPropertyItem instance.
 Delete (see page 63)	Deletes Item at index
 Destroy (see page 63)	Destroys an instance of TPropertyItem.
 Expandable (see page 63)	Specifies whether property item can be expanded.
 GetName (see page 63)	Returns name of the item. May be overridden in derived class.
 HasValue (see page 63)	Specifies whether property item has value. For example, category item does not have value.
 IndexOf (see page 63)	Returns index of child item. If Item is not a child returns -1.
 Insert (see page 63)	Adds a property item to the Items (see page 64) array at the position specified by Index.
 IsEqual (see page 63)	Returns True if property items are equal.
 IsRoot (see page 63)	Returns True if the item is root (see page 64) item.
 Move (see page 63)	Changes the position of an item in the Items (see page 64) array.
 Root (see page 64)	Specifies Root item.




**TPropListRoot Class**

TPropListRoot Class	Description
 BeginUpdate (see page 67)	Suspends updating of property list.
 Changed (see page 67)	Called when property item was changed.
 Create (see page 67)	Creates and initializes a TPropListRoot instance.
 Destroy (see page 68)	Destroys an instance of TPropListRoot.
 EndUpdate (see page 68)	Re-enables screen repainting.
 ExpandItem (see page 68)	Called when item is to be expanded.
 ExpIndexOf (see page 68)	Returns visible index (expanded) of specified property node.
 RestoreState (see page 68)	Restores previously saved state.
 SaveState (see page 68)	Saves state, i.e. expanded items and select item.
 UpdateList (see page 68)	Updates property list.

**TPropertyItem Properties**

TPropertyItem Properties	Description
 Count (see page 64)	Determines count of child items.
 DisplayName (see page 64)	Specifies name displayed on screen
 Expanded (see page 64)	Specifies if node is expanded or not.
 Items (see page 64)	Provides indexed access to the child items.
 Level (see page 64)	Indicates the level of indentation of a item within the property list control..
 Name (see page 64)	Specifies the name of the property node.
 Parent (see page 65)	Indicates the parent property of the node.
 PathName (see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
 Visible (see page 65)	Specifies whether item is selected.

**TPropListRoot Class**

TPropListRoot Class	Description
 ExpCount (see page 68)	Returns count of child nodes and their nodes recursively
 Expltems (see page 68)	Provides indexed access to the list of visible items.
 Owner (see page 69)	Specifies custom property list - owner of property items collection.

**1.10.1.4.1 TPropListRoot Methods****1.10.1.4.1.1 TPropListRoot.BeginUpdate Method**

Suspends updating of property list.

```
procedure BeginUpdate;
```

**Description**

The BeginUpdate method suspends screen repainting until the EndUpdate (see page 68) method is called. Use BeginUpdate to speed processing and avoid flicker while items are added to or deleted from a property list.

**1.10.1.4.1.2 TPropListRoot.Changed Method**

Called when property item was changed.

```
procedure Changed; override;
```

**1.10.1.4.1.3 TPropListRoot.Create Constructor**

Creates and initializes a TPropListRoot instance.

```
constructor Create(AOwner: TDualList);
```

**Description**

Use Create to programmatically instantiate a TPropListRoot object.



#### 1.10.1.4.1.4 TPropListRoot.Destroy Destructor

Destroys an instance of TPropListRoot.

```
destructor Destroy; override;
```

##### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

#### 1.10.1.4.1.5 TPropListRoot.EndUpdate Method

Re-enables screen repainting.

```
procedure EndUpdate;
```

##### Description

Use EndUpdate to re-enable screen repainting that was turned off with the BeginUpdate (see page 67) method.

#### 1.10.1.4.1.6 TPropListRoot.ExpandItem Method

Called when item is to be expanded.

```
procedure ExpandItem(Item: TPropertyItem); virtual;
```

#### 1.10.1.4.1.7 TPropListRoot.ExpIndexOf Method

Returns visible index (expanded) of specified property node.

```
function ExpIndexOf(Item: TPropertyItem): integer;
```

#### 1.10.1.4.1.8 TPropListRoot.RestoreState Method

Restores previously saved state.

```
procedure RestoreState;
```

#### 1.10.1.4.1.9 TPropListRoot.SaveState Method

Saves state, i.e. expanded items and select item.

```
procedure SaveState;
```

#### 1.10.1.4.1.10 TPropListRoot.UpdateList Method

Updates property list.

```
procedure UpdateList;
```

### 1.10.1.4.2 TPropListRoot Properties

#### 1.10.1.4.2.1 TPropListRoot.ExpCount Property

Returns count of child nodes and their nodes recursively

```
property ExpCount: integer;
```

##### Description

ExpCount recursively computes count of visible child nodes.

#### 1.10.1.4.2.2 TPropListRoot.ExplItems Property

Provides indexed access to the list of visible items.

```
property ExpItems [Index: integer]: TPropertyItem;
```

Description

ExplItems property provides indexed access not only to the children of the root (see page 64) item but to all visible (expanded) items.

1.10.1.4.2.3 TPropListRoot.Owner Property


Specifies custom property list - owner of property items collection.

```
property Owner: TDualList;
```

1.10.2 Structs, Records, Enums

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

Enumerations

Enumeration	Description
 TCellType (see page 69)	Specifies type of the cell in property list control.

Legend

	Enumeration
-----------------------------------------------------------------------------------	-------------

1.10.2.1 ecDIList.TCellType Enumeration

```
TCellType = (  
    ctPropName,  
    ctPropValue,  
    ctCategory  
);
```

File

ecDIList

Members

Members	Description
ctPropName	Cell is name cell, left cell in dual list.
ctPropValue	Cell is value cell, i.e. right cell in dual list.
ctCategory	Cell is category item, i.e. it occupies both name and value cells.

Description

Specifies type of the cell in property list control.

1.10.3 Types

The following table lists types in this documentation.

Types

Type	Description
TCustomPropDrawEvent (see page 70)	See TCustomPropList.OnDrawPropCell Event (see page 51)
TGetCellParamsEvent (see page 70)	See TCustomPropList.OnGetCellParams Event (see page 52)

### 1.10.3.1 ecDList.TCustomPropDrawEvent Type

See TCustomPropList.OnDrawPropCell Event ([see page 51](#))

```
TCustomPropDrawEvent = procedure (Sender: TObject; const R: TRect; CellType: TCellType;
Node: TPropertyItem; Alignment: TAlignment) of object;
```

#### File

ecDList

### 1.10.3.2 ecDList.TGetCellParamsEvent Type

See TCustomPropList.OnGetCellParams Event ([see page 52](#))

```
TGetCellParamsEvent = procedure (Sender: TObject; Node: TPropertyItem; Canvas: TCanvas;
CellType: TCellType; var Alignment: TAlignment) of object;
```

#### File

ecDList

## 1.11 ecExtEdit Namespace

### 1.11.1 Classes

The following table lists classes in this documentation.

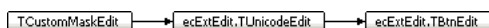
#### Classes

Class	Description
TBtnEdit ( <a href="#">see page 70</a> )	TBtnEdit is the base class from which TCustomEditEx ( <a href="#">see page 78</a> ) control is derived.
TCustomEditEx ( <a href="#">see page 78</a> )	TCustomEditEx is an extended editor with button.
TEditEx ( <a href="#">see page 85</a> )	TExtEdit is an extended editor with button.
TPopupListbox ( <a href="#">see page 105</a> )	TPopupListbox represents drop-down list in TExtEdit type.
TUnicodeEdit ( <a href="#">see page 107</a> )	Mask edit control with Unicode support.

#### 1.11.1.1 TBtnEdit Class

TBtnEdit is the base class from which TCustomEditEx ([see page 78](#)) control is derived.

#### Class Hierarchy



```
TBtnEdit = class(TUnicodeEdit);
```

#### File

ecExtEdit

#### Description

TBtnEdit is inherited from TUnicodeEdit ([see page 107](#)) class.




















It introduces optional button at the right side and optional status area at the left side of edit control

## Members





### TUnicodeEdit Methods

TUnicodeEdit Methods	Description
 Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
 Destroy (see page 108)	Destroys an instance of TUnicodeEdit.










### TBtEdit Class

TBtEdit Class	Description
 AdjustClientRect (see page 73)	Overrides the inherited method.
 ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
 Create (see page 73)	Creates and initializes a TBtEdit instance.
 CreateParams (see page 73)	Overrides the inherited method.
 CreateWnd (see page 73)	Overrides the inherited method.
 Destroy (see page 74)	Destroys an instance of TBtEdit
 EndTracking (see page 74)	Called after finishing the mouse tracking
 KeyDown (see page 74)	Overrides the inherited method.
 KeyPress (see page 74)	Overrides the inherited method.
 MouseMove (see page 74)	Overrides the inherited method.
 MouseUp (see page 75)	Overrides the inherited method.
 Paint (see page 75)	Overrides the base rendering method.
 PaintBtnGlyph (see page 75)	Renders the image of the button.
 PaintStatus (see page 75)	Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.
 PaintWindow (see page 75)	Overrides the inherited method.
 PtInButton (see page 75)	Checks if specified point is over the button
 StartTracking (see page 76)	Calls immediately after user pushes the button.
 StopTracking (see page 76)	Calls immediately after user releases the button.
 TrackButton (see page 76)	Controls tracking button process.

### TUnicodeEdit Properties

TUnicodeEdit Properties	Description
 IsUnicode (see page 108)	Specifies whether control is Unicode edit.
 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).

### TBtEdit Class







TBtEdit Class	Description
 Alignment (see page 76)	Determines how the text is aligned within the editor control.
 ButtonVisible (see page 76)	Specifies if button-like rectangle at the right edge of the control is visible.
 ButtonWidth (see page 77)	Specifies width in pixels of the button.
 Canvas (see page 77)	Provides access to the drawing surface of the TBtEdit.
 MultiLine (see page 77)	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth (see page 77)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
 WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

### TBtEdit Events

#### TBtEdit Class

TBtEdit Class	Description
 OnButtonClick (see page 78)	Occurs when the user clicks the button.

**Legend**

	Constructor
	virtual
	protected
	Property
	read only
	Event




















**TBtEdit Events****TBtEdit Class**

TBtEdit Class	Description
 OnButtonClick ( <a href="#">see page 78</a> )	Occurs when the user clicks the button.





**TUnicodeEdit Methods**

TUnicodeEdit Methods	Description
 Create ( <a href="#">see page 108</a> )	Creates and initializes a TUnicodeEdit instance.
 Destroy ( <a href="#">see page 108</a> )	Destroys an instance of TUnicodeEdit.









**TBtEdit Class**


TBtEdit Class	Description
 AdjustClientRect ( <a href="#">see page 73</a> )	Overrides the inherited method.
 ButtonClick ( <a href="#">see page 73</a> )	Simulates a button click, as if the user had clicked the button.
 Create ( <a href="#">see page 73</a> )	Creates and initializes a TBtEdit instance.
 CreateParams ( <a href="#">see page 73</a> )	Overrides the inherited method.
 CreateWnd ( <a href="#">see page 73</a> )	Overrides the inherited method.
 Destroy ( <a href="#">see page 74</a> )	Destroys an instance of TBtEdit
 EndTracking ( <a href="#">see page 74</a> )	Called after finishing the mouse tracking
 KeyDown ( <a href="#">see page 74</a> )	Overrides the inherited method.
 KeyPress ( <a href="#">see page 74</a> )	Overrides the inherited method.
 MouseMove ( <a href="#">see page 74</a> )	Overrides the inherited method.
 MouseUp ( <a href="#">see page 75</a> )	Overrides the inherited method.
 Paint ( <a href="#">see page 75</a> )	Overrides the base rendering method.
 PaintBtnGlyph ( <a href="#">see page 75</a> )	Renders the image of the button.
 PaintStatus ( <a href="#">see page 75</a> )	Paints status area. This method is called only if StatusWidth ( <a href="#">see page 77</a> ) is greater 0.
 PaintWindow ( <a href="#">see page 75</a> )	Overrides the inherited method.
 PtInButton ( <a href="#">see page 75</a> )	Checks if specified point is over the button
 StartTracking ( <a href="#">see page 76</a> )	Calls immediately after user pushes the button.
 StopTracking ( <a href="#">see page 76</a> )	Calls immediately after user releases the button.
 TrackButton ( <a href="#">see page 76</a> )	Controls tracking button process.

**TUnicodeEdit Properties**

TUnicodeEdit Properties	Description
 IsUnicode ( <a href="#">see page 108</a> )	Specifies whether control is Unicode edit.
 SelTextW ( <a href="#">see page 109</a> )	Specifies the selected portion of the edit control's text (Unicode version).
 Text ( <a href="#">see page 109</a> )	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW ( <a href="#">see page 109</a> )	Specifies the text string that is displayed in the edit box (Ansi version).

**TBtEdit Class**

TBtEdit Class	Description
 Alignment ( <a href="#">see page 76</a> )	Determines how the text is aligned within the editor control.
 ButtonVisible ( <a href="#">see page 76</a> )	Specifies if button-like rectangle at the right edge of the control is visible.
 ButtonWidth ( <a href="#">see page 77</a> )	Specifies width in pixels of the button.
 Canvas ( <a href="#">see page 77</a> )	Provides access to the drawing surface of the TBtEdit.
 MultiLine ( <a href="#">see page 77</a> )	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth ( <a href="#">see page 77</a> )	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns ( <a href="#">see page 77</a> )	Determines whether the user can insert return characters into the text.
 WantTabs ( <a href="#">see page 78</a> )	Determines whether the user can insert tab characters into the text.

 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.
----------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------

## 1.11.1.1.1 TBtnEdit Methods

### 1.11.1.1.1.1 TBtnEdit.AdjustClientRect Method

Overrides the inherited method.

```
procedure AdjustClientRect(var Rect: TRect); override;
```

#### Description

First TBtnEdit (see page 70) calls inherited method then subtracts width of button from the right side of Rect parameter.

### 1.11.1.1.1.2 TBtnEdit.ButtonClick Method

Simulates a button click, as if the user had clicked the button.

```
procedure ButtonClick; virtual;
```

#### Description

Calling ButtonClick generates an OnButtonClick (see page 78) event.

### 1.11.1.1.1.3 TBtnEdit.Create Constructor

Creates and initializes a TBtnEdit (see page 70) instance.

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

#### Description

Use Create to programmatically instantiate this type of a control.

#### Create

- Calls the inherited Create method
- Sets the width of the button calling GetSystemMetrics method with SM\_CXVSCROLL parameter
- Sets ButtonVisible (see page 76) to false
- Sets Alignment (see page 76) to taLeftJustify
- Sets MultiLine (see page 77) to true
- Creates Canvas (see page 77) object and sets its Control property to the control itself.

### 1.11.1.1.1.4 TBtnEdit.CreateParams Method

Overrides the inherited method.

```
procedure CreateParams(var Params: TCreateParams); override;
```

#### Description

It initializes a window-creation parameter record passed in the Params parameter.

#### CreateParams

- Calls inherited method
- Adds to Style property Alignment (see page 76) and Multiline value

### 1.11.1.1.1.5 TBtnEdit.CreateWnd Method

Overrides the inherited method.

```
procedure CreateWnd; override;
```

**Description****CreateWnd**

- Calls inherited method
- Updates button position

**1.11.1.1.1.6 TBtnEdit.Destroy Destructor**

Destroys an instance of TBtnEdit (see page 70)

```
destructor Destroy; override;
```

**Description**

As the control is destroyed, it:

- Destroys the TCanvas object in its Canvas (see page 77) property.
- Calls the inherited Destroy method

**1.11.1.1.1.7 TBtnEdit.EndTracking Method**

Called after finishing the mouse tracking

```
procedure EndTracking(Pressed: Boolean); virtual;
```

**Description**

Calls in the MouseUp (see page 75) method. If Pressed property is True it calls ButtonClick (see page 73) method.

**1.11.1.1.1.8 TBtnEdit.KeyDown Method**

Overrides the inherited method.

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

**Description**

If Key parameter is VK\_RETURN and Shift is equal to ssCtrl it emulates ButtonClick (see page 73). Otherwise it calls inherited method.

**1.11.1.1.1.9 TBtnEdit.KeyPress Method**

Overrides the inherited method.

```
procedure KeyPress(var Key: Char); override;
```

**Description**

If Key parameter is VK\_ESCAPE or VK\_RETURN it

- notify parent form
- if Key is equal to VK\_RETURN then calls inherited method and cleanse Key

Else it just calls inherited method.

**1.11.1.1.1.10 TBtnEdit.MouseMove Method**

Overrides the inherited method.

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

**Description**

First it calls TrackButton (see page 76) method if in tracking mode.

Then it calls inherited method.

#### 1.11.1.1.1.11 TBtnEdit.MouseUp Method

Overrides the inherited method.

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
override;
```

##### Description

First it stops tracking process (if any).

Then it calls inherited method.

#### 1.11.1.1.1.12 TBtnEdit.Paint Method

Overrides the base rendering method.

```
procedure Paint; virtual;
```

##### Description

Does nothing in the TBtnEdt.

It uses in PaintWindow (see page 75) as part of template method.

Introduced for overriding in the descendants.

#### 1.11.1.1.1.13 TBtnEdit.PaintBtnGlyph Method

Renders the image of the button.

```
procedure PaintBtnGlyph(Canvas: TCanvas; Rect: TRect); virtual;
```

##### Description

Does nothing in the TBtnEdt.

It uses in PaintWindow (see page 75) as part of template method.

Introduced for overriding in the descendants.

#### 1.11.1.1.1.14 TBtnEdit.PaintStatus Method

Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.

```
procedure PaintStatus(Canvas: TCanvas; Rect: TRect); virtual;
```

#### 1.11.1.1.1.15 TBtnEdit.PaintWindow Method

Overrides the inherited method.

```
procedure PaintWindow(DC: HDC); override;
```

##### Description

First it renders button if button is visible.

- Draws edge
- Calls PaintBtnGlyph (see page 75)

Second it updates text flags and calls Paint (see page 75) method.

#### 1.11.1.1.1.16 TBtnEdit.PtInButton Method

Checks if specified point is over the button

```
function PtInButton(p: TPoint): Boolean;
```



**Description**

It uses in WM\_LBUTTONDOWN, WM\_LBUTTONDBLCLK, WM\_SETCURSOR messages and TrackButton (see page 76) method to verify if specified point is over the button rectangle.

**1.11.1.1.1.17 TBtnEdit.StartTracking Method**

Calls immediately after user pushes the button.

```
procedure StartTracking; virtual;
```

**Description**

Stops tracking button and sets MouseCapture property to False.

**1.11.1.1.1.18 TBtnEdit.StopTracking Method**

Calls immediately after user releases the button.

```
procedure StopTracking;
```

**Description**

Stops tracking button and sets MouseCapture property to False.

**1.11.1.1.1.19 TBtnEdit.TrackButton Method**

Controls tracking button process.

```
procedure TrackButton(X: Integer; Y: Integer);
```

**Description**

- Determines if point with such coordinates is over button.
- Repaints the button area

**1.11.1.1.2 TBtnEdit Properties****1.11.1.1.2.1 TBtnEdit.Alignment Property**

Determines how the text is aligned within the editor control.

```
property Alignment: TAlignment;
```

**Description**

Use Alignment to change the way the text is formatted by the in-place editor control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control
taRightJustify	Align text to the right side of the control

**1.11.1.1.2.2 TBtnEdit.ButtonVisible Property**

Specifies if button-like rectangle at the right edge of the control is visible.

```
property ButtonVisible: Boolean;
```

**Description**

There are three variants of TBtnEdit (see page 70) control appearance

EditStyle	Appearance
ieSimple	No button at all
ieEllipsis	Button has ellipsis
iePickList	Button with drop-down list (has arrow)

There are two ways to change this property: direct setting to True or False or setting EditStyle property.

#### 1.11.1.1.2.3 TBtnEdit.ButtonWidth Property

Specifies width in pixels of the button.

```
property ButtonWidth: integer;
```

##### Description

This property specifies width in pixels of the button if any.

Use this property to read or set width of the button.

#### 1.11.1.1.2.4 TBtnEdit.Canvas Property

Provides access to the drawing surface of the TBtnEdit (see page 70).

```
property Canvas: TCanvas;
```

##### Description

Use Canvas as the drawing surface when customizing the way the control paints itself. For example, when using a TBtnEdit descendant there is a way to use Canvas to make specific drawing action.

#### 1.11.1.1.2.5 TBtnEdit.MultiLine Property

Designates a multiline edit control. The default is single-line edit control.

```
property MultiLine: Boolean;
```

##### Description

When MultiLine is True TBtnEdit (see page 70) is equivalent to TMemo control, otherwise it is equivalent to TEdit control.

#### 1.11.1.1.2.6 TBtnEdit.StatusWidth Property

Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.

```
property StatusWidth: integer;
```

#### 1.11.1.1.2.7 TBtnEdit.WantReturns Property

Determines whether the user can insert return characters into the text.

```
property WantReturns: Boolean;
```

##### Description

Set WantReturns to true to allow users to enter return characters into the text. Set WantReturns to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if WantReturns is false, pressing Enter chooses the default button. If WantReturns is true, pressing Enter inserts a return character in the text.

##### Notes

If WantReturns is false, users can still enter return characters into the text by pressing Ctrl+Enter.

### 1.11.1.1.2.8 TBtnEdit.WantTabs Property

Determines whether the user can insert tab characters into the text.

```
property WantTabs: Boolean;
```

#### Description

Set WantTabs to true to allow users to enter tab characters into the text. Set WantTabs to false if you want the tab character to select the next control on the form instead.

#### Notes

If WantTabs is true, users can tab into the edit control, but they can't tab out.

### 1.11.1.1.2.9 TBtnEdit.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

```
property WordWrap: Boolean;
```

#### Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

#### Notes

There should be no use for a horizontal scroll bar if WordWrap is true.

## 1.11.1.1.3 TBtnEdit Events

### 1.11.1.1.3.1 TBtnEdit.OnButtonClick Event

Occurs when the user clicks the button.

```
property OnButtonClick: TNotifyEvent;
```

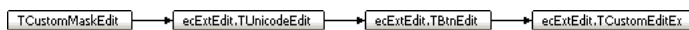
#### Description

Use the OnButtonClick event handler to respond when the user clicks the button or do some equivalent action such as pressing the associated short keys or some. This is very similar to standard OnClick event.

## 1.11.1.2 TCustomEditEx Class

TCustomEditEx is an extended editor with button.

#### Class Hierarchy



```
TCustomEditEx = class(TBtnEdit);
```

#### File

ecExtEdit

#### Description

TCustomEditEx is an extended editor with button that may have one of three styles:

- Button with ellipsis. Represents possibility to call some dialog when pressed.
- Button with a scrollable list. Identical to TComboBox type.
- Simple editor without button




















This type is used as in-place editors representation in Object Inspector.

## Members














### TUnicodeEdit Methods

TUnicodeEdit Methods	Description
 Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
 Destroy (see page 108)	Destroys an instance of TUnicodeEdit.





### TBttnEdit Class

TBttnEdit Class	Description
 AdjustClientRect (see page 73)	Overrides the inherited method.
 ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
 Create (see page 73)	Creates and initializes a TBttnEdit (see page 70) instance.
 CreateParams (see page 73)	Overrides the inherited method.
 CreateWnd (see page 73)	Overrides the inherited method.
 Destroy (see page 74)	Destroys an instance of TBttnEdit (see page 70)
 EndTracking (see page 74)	Called after finishing the mouse tracking
 KeyDown (see page 74)	Overrides the inherited method.
 KeyPress (see page 74)	Overrides the inherited method.
 MouseMove (see page 74)	Overrides the inherited method.
 MouseUp (see page 75)	Overrides the inherited method.
 Paint (see page 75)	Overrides the base rendering method.
 PaintBtnGlyph (see page 75)	Renders the image of the button.
 PaintStatus (see page 75)	Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.
 PaintWindow (see page 75)	Overrides the inherited method.
 PtInButton (see page 75)	Checks if specified point is over the button
 StartTracking (see page 76)	Calls immediately after user pushes the button.
 StopTracking (see page 76)	Calls immediately after user releases the button.
 TrackButton (see page 76)	Controls tracking button process.

### TCustomEditEx Class









TCustomEditEx Class	Description
 AcceptListValue (see page 82)	Active pop-up window.
 ButtonClick (see page 82)	Simulates a button click, as if the user had clicked the button.
 CloseUp (see page 82)	Generates an OnCloseUp (see page 85) event and makes some other actions.
 Create (see page 82)	Creates and initializes a TBttnEdit instance.
 Destroy (see page 82)	Destroys an instance of TBttnEdit
 DoDropDownKeys (see page 83)	Works as a method dispatcher depending on parameter values.
 DropDown (see page 83)	Generates an OnDropDown (see page 85) event.
 EndTracking (see page 83)	Called after finishing the mouse tracking
 KeyPress (see page 83)	Respond to keyboard input.
 MouseDown (see page 83)	Overrides inherited method
 MouseMove (see page 83)	Overrides the inherited method.
 PaintBtnGlyph (see page 84)	Overrides inherited method
 StartTracking (see page 84)	Overrides inherited property

### TUnicodeEdit Properties






TUnicodeEdit Properties	Description
 IsUnicode (see page 108)	Specifies whether control is Unicode edit.
 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).

### TBttnEdit Class

TBttnEdit Class	Description
 Alignment (see page 76)	Determines how the text is aligned within the editor control.
 ButtonVisible (see page 76)	Specifies if button-like rectangle at the right edge of the control is visible.

 ButtonWidth (see page 77)	Specifies width in pixels of the button.
  Canvas (see page 77)	Provides access to the drawing surface of the TBtnEdit (see page 70).
 MultiLine (see page 77)	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth (see page 77)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
 WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

### TCustomEditEx Class





TCustomEditEx Class	Description
 ActiveList (see page 84)	Specifies current popup control.
 EditStyle (see page 84)	Specifies edit style of the button
 ListAlign (see page 84)	Specifies relative align of popup control.
  PickList (see page 84)	Determines the drop-down list.

### TBtnEdit Events







#### TBtnEdit Class

TBtnEdit Class	Description
 OnButtonClick (see page 78)	Occurs when the user clicks the button.

### TCustomEditEx Class

TCustomEditEx Class	Description
 OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
 OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
 OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
 OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.

### Legend





	Constructor
	virtual
	protected
	Property
	read only
	Event

### TBtnEdit Events

#### TBtnEdit Class

TBtnEdit Class	Description
 OnButtonClick (see page 78)	Occurs when the user clicks the button.






### TCustomEditEx Class

















TCustomEditEx Class	Description
 OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
 OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
 OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
 OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.

### TUnicodeEdit Methods














TUnicodeEdit Methods	Description
  Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
  Destroy (see page 108)	Destroys an instance of TUnicodeEdit.

### TBtnEdit Class





TBtnEdit Class	Description
  AdjustClientRect (see page 73)	Overrides the inherited method.
  ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
 Create (see page 73)	Creates and initializes a TBtnEdit (see page 70) instance.

 <b>CreateParams</b> ( <a href="#">see page 73</a> )	Overrides the inherited method.
 <b>CreateWnd</b> ( <a href="#">see page 73</a> )	Overrides the inherited method.
 <b>Destroy</b> ( <a href="#">see page 74</a> )	Destroys an instance of TBtnEdit ( <a href="#">see page 70</a> )
 <b>EndTracking</b> ( <a href="#">see page 74</a> )	Called after finishing the mouse tracking
 <b>KeyDown</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
 <b>KeyPress</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
 <b>MouseMove</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
 <b>MouseUp</b> ( <a href="#">see page 75</a> )	Overrides the inherited method.
 <b>Paint</b> ( <a href="#">see page 75</a> )	Overrides the base rendering method.
 <b>PaintBtnGlyph</b> ( <a href="#">see page 75</a> )	Renders the image of the button.
 <b>PaintStatus</b> ( <a href="#">see page 75</a> )	Paints status area. This method is called only if StatusWidth ( <a href="#">see page 77</a> ) is greater 0.
 <b>PaintWindow</b> ( <a href="#">see page 75</a> )	Overrides the inherited method.
 <b>PtInButton</b> ( <a href="#">see page 75</a> )	Checks if specified point is over the button
 <b>StartTracking</b> ( <a href="#">see page 76</a> )	Calls immediately after user pushes the button.
 <b>StopTracking</b> ( <a href="#">see page 76</a> )	Calls immediately after user releases the button.
 <b>TrackButton</b> ( <a href="#">see page 76</a> )	Controls tracking button process.










### TCustomEditEx Class

TCustomEditEx Class	Description
 <b>AcceptListValue</b> ( <a href="#">see page 82</a> )	Active pop-up window.
 <b>ButtonClick</b> ( <a href="#">see page 82</a> )	Simulates a button click, as if the user had clicked the button.
 <b>CloseUp</b> ( <a href="#">see page 82</a> )	Generates an OnCloseUp ( <a href="#">see page 85</a> ) event and makes some other actions.
 <b>Create</b> ( <a href="#">see page 82</a> )	Creates and initializes a TBtnEdit instance.
 <b>Destroy</b> ( <a href="#">see page 82</a> )	Destroys an instance of TBtnEdit
 <b>DoDropDownKeys</b> ( <a href="#">see page 83</a> )	Works as a method dispatcher depending on parameter values.
 <b>DropDown</b> ( <a href="#">see page 83</a> )	Generates an OnDropDown ( <a href="#">see page 85</a> ) event.
 <b>EndTracking</b> ( <a href="#">see page 83</a> )	Called after finishing the mouse tracking
 <b>KeyPress</b> ( <a href="#">see page 83</a> )	Respond to keyboard input.
 <b>MouseDown</b> ( <a href="#">see page 83</a> )	Overrides inherited method
 <b>MouseMove</b> ( <a href="#">see page 83</a> )	Overrides the inherited method.
 <b>PaintBtnGlyph</b> ( <a href="#">see page 84</a> )	Overrides inherited method
 <b>StartTracking</b> ( <a href="#">see page 84</a> )	Overrides inherited property



### TUnicodeEdit Properties


TUnicodeEdit Properties	Description
 <b>IsUnicode</b> ( <a href="#">see page 108</a> )	Specifies whether control is Unicode edit.
 <b>SelTextW</b> ( <a href="#">see page 109</a> )	Specifies the selected portion of the edit control's text (Unicode version).
 <b>Text</b> ( <a href="#">see page 109</a> )	Specifies the text string that is displayed in the edit box (Ansi version).
 <b>TextW</b> ( <a href="#">see page 109</a> )	Specifies the text string that is displayed in the edit box (Ansi version).

### TBtnEdit Class

TBtnEdit Class	Description
 <b>Alignment</b> ( <a href="#">see page 76</a> )	Determines how the text is aligned within the editor control.
 <b>ButtonVisible</b> ( <a href="#">see page 76</a> )	Specifies if button-like rectangle at the right edge of the control is visible.
 <b>ButtonWidth</b> ( <a href="#">see page 77</a> )	Specifies width in pixels of the button.
 <b>Canvas</b> ( <a href="#">see page 77</a> )	Provides access to the drawing surface of the TBtnEdit ( <a href="#">see page 70</a> ).
 <b>MultiLine</b> ( <a href="#">see page 77</a> )	Designates a multiline edit control. The default is single-line edit control.
 <b>StatusWidth</b> ( <a href="#">see page 77</a> )	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 <b>WantReturns</b> ( <a href="#">see page 77</a> )	Determines whether the user can insert return characters into the text.
 <b>WantTabs</b> ( <a href="#">see page 78</a> )	Determines whether the user can insert tab characters into the text.
 <b>WordWrap</b> ( <a href="#">see page 78</a> )	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

### TCustomEditEx Class

TCustomEditEx Class	Description
 <b>ActiveList</b> ( <a href="#">see page 84</a> )	Specifies current popup control.
 <b>EditStyle</b> ( <a href="#">see page 84</a> )	Specifies edit style of the button

 ListAlign (see page 84)	Specifies relative align of popup control.
 PickList (see page 84)	Determines the drop-down list.

## 1.11.1.2.1 TCustomEditEx Methods

### 1.11.1.2.1.1 TCustomEditEx.AcceptListValue Method

Active pop-up window.

```
procedure AcceptListValue(var ListValue: string); virtual;
```

#### Description

This is an active pop-up window.

It is used when EditStyle (see page 84) is iePickList and drop-down list is open.

### 1.11.1.2.1.2 TCustomEditEx.ButtonClick Method

Simulates a button click, as if the user had clicked the button.

```
procedure ButtonClick; override;
```

#### Description

Calling ButtonClick generates an OnButtonClick event.

### 1.11.1.2.1.3 TCustomEditEx.CloseUp Method

Generates an OnCloseUp (see page 85) event and makes some other actions.

```
procedure CloseUp(Accept: Boolean);
```

#### Description

This method generates an OnCloseUp (see page 85) event and calls AcceptListValue (see page 82) if Accept and EditCanModify are both true.

### 1.11.1.2.1.4 TCustomEditEx.Create Constructor

Creates and initializes a TBtnEdit instance.

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

#### Description

Use Create to programmatically instantiate this type of a control.

#### Create

- Calls the inherited Create method
- Sets the width of the button calling GetSystemMetrics method with SM\_CXVSCROLL parameter
- Sets ButtonVisible to false
- Sets Alignment to taLeftJustify
- Sets MultiLine to true
- Creates Canvas object and sets its Control property to the control itself.

### 1.11.1.2.1.5 TCustomEditEx.Destroy Destructor

Destroys an instance of TBtnEdit

```
destructor Destroy; override;
```

**Description**

As the control is destroyed, it:

- Destroys the TCanvas object in its Canvas property.
- Calls the inherited Destroy method

**1.11.1.2.1.6 TCustomEditEx.DoDropDownKeys Method**

Works as a method dispatcher depending on parameter values.

```
procedure DoDropDownKeys(var Key: Word; Shift: TShiftState);
```

**Description**

This method calls another depending on some circumstances

- if Key is equal to VK\_UP or VK\_DOWN and Shift contains ssAlt it drops or closes the drop-down list up
- if Key is equal to VK\_RETURN or VK\_ESCAPE and Shift does not contain ssAlt it closes the drop-down list up

**1.11.1.2.1.7 TCustomEditEx.DropDown Method**

Generates an OnDropDown (see page 85) event.

```
procedure DropDown; dynamic;
```

**Description**

This method prepares and drops the drop-down list (if any).

In addition it generates OnDropDown (see page 85) event.

**1.11.1.2.1.8 TCustomEditEx.EndTracking Method**

Called after finishing the mouse tracking

```
procedure EndTracking(Pressed: Boolean); override;
```

**Description**

Calls in the MouseUp method. If Pressed property is True it calls ButtonClick method.

**1.11.1.2.1.9 TCustomEditEx.KeyPress Method**

Respond to keyboard input.

```
procedure KeyPress(var Key: Char); override;
```

**Description**

Closing drop-down list up (if any and visible) when Key is equal to VK\_RETURN and VK\_ESCAPE then calls inherited method.

**1.11.1.2.1.10 TCustomEditEx.MouseDown Method**

Overrides inherited method

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
override;
```

**Description**

Closes drop-down list up and calls inherited method.

**1.11.1.2.1.11 TCustomEditEx.MouseMove Method**

Overrides the inherited method.



```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

**Description**

First it calls TrackButton method if in tracking mode.

Then it calls inherited method.

**1.11.1.2.1.12 TCustomEditEx.PaintBtnGlyph Method**

Overrides inherited method

```
procedure PaintBtnGlyph(Canvas: TCanvas; Rect: TRect); override;
```

**Description**

Paints glyphs on the button surface depending on EditStyle (see page 84) property.

**1.11.1.2.1.13 TCustomEditEx.StartTracking Method**

Overrides inherited property

```
procedure StartTracking; override;
```

**Description**

Calls inherited method, then closes up or drops down drop-down list.

**1.11.1.2.2 TCustomEditEx Properties****1.11.1.2.2.1 TCustomEditEx.ActiveList Property**

Specifies current popup control.

```
property ActiveList: TWinControl;
```

**Description**

When popup is not shown ActiveList is nil otherwise it is equal to PickList (see page 84).

**1.11.1.2.2.2 TCustomEditEx.EditStyle Property**

Specifies edit style of the button

```
property EditStyle: TInplaceEditStyle;
```

**Description**

Specifies of what visual representation the button will be.

There are three styles available

- ieSimple - without button
- ieEllipsis - button with ellipsis
- iePickList - button with drop-down list

**1.11.1.2.2.3 TCustomEditEx.ListAlign Property**

Specifies relative align of popup control.

```
property ListAlign: TAlignment;
```

**1.11.1.2.2.4 TCustomEditEx.PickList Property**

Determines the drop-down list.

```
property PickList: TPopupListbox;
```

**Description**

Use this property to access drop-down list object for iePickList EditStyle (see page 84).

**1.11.1.2.3 TCustomEditEx Events****1.11.1.2.3.1 TCustomEditEx.OnAcceptListValue Event**

Occurs when the user makes right choice in the drop-down list.

**property** OnAcceptListValue: TOnAcceptListValueEvent;

**Description**

Write an OnAcceptListValue event handler to implement special processing that needs to occur when the user makes a choice in the drop-down list. For example, an OnAcceptListValue event handler can check whether the user can make or not such choice.

**1.11.1.2.3.2 TCustomEditEx.OnCloseUp Event**

Occurs when the drop-down list closes up due to some user action.

**property** OnCloseUp: TCloseUpEvent;

**Description**

Write an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

**1.11.1.2.3.3 TCustomEditEx.OnDropDown Event**

Occurs when the user opens the drop-down list.

**property** OnDropDown: TNotifyEvent;

**Description**

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

**1.11.1.2.3.4 TCustomEditEx.OnMeasureWidth Event**

Occurs when width of controls needs to be calculated.

**property** OnMeasureWidth: TMeasureWidthEvent;

**Description**

Parameter AWidth represents calculated width, in pixels, of specified in Value text for specified Canvas (see page 77).

Write an OnMeasureWidth event handler to specify the new user width.

**1.11.1.3 TEditEx Class**

TExtEdit is an extended editor with button.

**Class Hierarchy**

```
TEditEx = class(TCustomEditEx);
```

**File**

ecExtEdit

**Description**

TExtEdit is an extended editor with button that may have one of three styles:




















- Button with ellipsis. Represents possibility to call some dialog when pressed.
- Button with a scrollable list. Identical to TComboBox type.
- Simple editor without button

This type is used as in-place editors representation in Object Inspector.














**Members****TUnicodeEdit Methods**

TUnicodeEdit Methods	Description
 Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
 Destroy (see page 108)	Destroys an instance of TUnicodeEdit.





**TBtnEdit Class**

TBtnEdit Class	Description
 AdjustClientRect (see page 73)	Overrides the inherited method.
 ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
 Create (see page 73)	Creates and initializes a TBtnEdit (see page 70) instance.
 CreateParams (see page 73)	Overrides the inherited method.
 CreateWnd (see page 73)	Overrides the inherited method.
 Destroy (see page 74)	Destroys an instance of TBtnEdit (see page 70)
 EndTracking (see page 74)	Called after finishing the mouse tracking
 KeyDown (see page 74)	Overrides the inherited method.
 KeyPress (see page 74)	Overrides the inherited method.
 MouseMove (see page 74)	Overrides the inherited method.
 MouseUp (see page 75)	Overrides the inherited method.
 Paint (see page 75)	Overrides the base rendering method.
 PaintBtnGlyph (see page 75)	Renders the image of the button.
 PaintStatus (see page 75)	Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.
 PaintWindow (see page 75)	Overrides the inherited method.
 PtInButton (see page 75)	Checks if specified point is over the button
 StartTracking (see page 76)	Calls immediately after user pushes the button.
 StopTracking (see page 76)	Calls immediately after user releases the button.
 TrackButton (see page 76)	Controls tracking button process.











**TCustomEditEx Class**

TCustomEditEx Class	Description
 AcceptListValue (see page 82)	Active pop-up window.
 ButtonClick (see page 82)	Simulates a button click, as if the user had clicked the button.
 CloseUp (see page 82)	Generates an OnCloseUp (see page 85) event and makes some other actions.
 Create (see page 82)	Creates and initializes a TBtnEdit instance.
 Destroy (see page 82)	Destroys an instance of TBtnEdit
 DoDropDownKeys (see page 83)	Works as a method dispatcher depending on parameter values.
 DropDown (see page 83)	Generates an OnDropDown (see page 85) event.
 EndTracking (see page 83)	Called after finishing the mouse tracking
 KeyPress (see page 83)	Respond to keyboard input.
 MouseDown (see page 83)	Overrides inherited method
 MouseMove (see page 83)	Overrides the inherited method.
 PaintBtnGlyph (see page 84)	Overrides inherited method
 StartTracking (see page 84)	Overrides inherited property






**TUnicodeEdit Properties**

<b>TUnicodeEdit Properties</b>	<b>Description</b>
 IsUnicode (see page 108)	Specifies whether control is Unicode edit.
 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).







**TBtnEdit Class**










































<b>TBtnEdit Class</b>	<b>Description</b>
 Alignment (see page 76)	Determines how the text is aligned within the editor control.
 ButtonVisible (see page 76)	Specifies if button-like rectangle at the right edge of the control is visible.
 ButtonWidth (see page 77)	Specifies width in pixels of the button.
  Canvas (see page 77)	Provides access to the drawing surface of the TBtnEdit (see page 70).
 MultiLine (see page 77)	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth (see page 77)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
 WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

**TCustomEditEx Class**

<b>TCustomEditEx Class</b>	<b>Description</b>
 ActiveList (see page 84)	Specifies current popup control.
 EditStyle (see page 84)	Specifies edit style of the button
 ListAlign (see page 84)	Specifies relative align of popup control.
  PickList (see page 84)	Determines the drop-down list.

**TEditEx Class**

<b>TEditEx Class</b>	<b>Description</b>
 Alignment (see page 92)	Determines how the text is aligned within the editor control.
 Anchors (see page 92)	Specifies how the control is anchored to its parent.
 AutoSelect (see page 92)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
 AutoSize (see page 92)	Determines whether the height of the edit control automatically resizes to accommodate the text.
 BevelEdges (see page 93)	Specifies which edges of the control are beveled.
 BevelInner (see page 93)	Specifies the cut of the inner bevel.
 BevelKind (see page 93)	Specifies the control's bevel style.
 BevelOuter (see page 93)	Specifies the cut of the outer bevel.
 BevelWidth (see page 93)	Specifies the width of the inner and outer bevels.
 BiDiMode (see page 94)	Specifies the bi-directional mode for the control.
 BorderStyle (see page 94)	Determines the style of the line drawn around the perimeter of the control.
 ButtonWidth (see page 94)	Specifies width in pixels of the button.
 CharCase (see page 94)	Determines the case of the text within the edit control.
 Color (see page 95)	Specifies the background color of the control.
 Constraints (see page 95)	Specifies the size constraints for the control.
 Ctl3D (see page 95)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DragCursor (see page 95)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 95)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 95)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 EditMask (see page 96)	Specifies the mask that represents what text is valid for the masked edit control.
 EditStyle (see page 96)	Specifies edit style of the button
 Enabled (see page 96)	Controls whether the control responds to mouse, keyboard, and timer events.
 Font (see page 96)	Controls the attributes of text written on or in the control.
 ImeMode (see page 96)	Determines the behavior of the input method editor (IME).

 ImeName (see page 97)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
 IsUnicode (see page 97)	Specifies whether control is Unicode edit.
 ListAlign (see page 97)	Specifies relative align of popup control.
 MaxLength (see page 97)	Specifies the maximum number of characters that can appear in the edit control.
 OnAcceptListValue (see page 98)	Occurs when the user makes right choice in the drop-down list.
 OnButtonClick (see page 98)	Occurs when the user clicks the button.
 OnChange (see page 98)	Occurs when the text for the edit control may have changed.
 OnClick (see page 98)	Occurs when the user clicks the control.
 OnCloseUp (see page 98)	Occurs when the drop-down list closes up due to some user action.
 OnDbClick (see page 98)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 98)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 99)	Occurs when the user drags an object over a control.
 OnDropDown (see page 99)	Occurs when the user opens the drop-down list.
 OnEndDrag (see page 99)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 99)	Occurs when a control receives the input focus.
 OnExit (see page 100)	Occurs when the input focus shifts away from one control to another.
 OnKeyDown (see page 100)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 100)	Occurs when key pressed.
 OnKeyUp (see page 100)	Occurs when the user releases a key that has been pressed.
 OnMeasureWidth (see page 101)	Occurs when width of controls needs to be calculated.
 OnMouseDown (see page 101)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 101)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 101)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnStartDrag (see page 102)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 102)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 102)	Determines where a control looks for its color information.
 ParentCtl3D (see page 102)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 102)	Determines where a control looks for its font information.
 ParentShowHint (see page 102)	Determines where a control looks to find out if its Help Hint should be shown.
 PasswordChar (see page 103)	Indicates the character, if any, to display in place of the actual characters typed in the control.
 PickList (see page 103)	Determines the drop-down list.
 PopupMenu (see page 103)	Identifies the pop-up menu associated with the control.
 ReadOnly (see page 103)	Determines whether the user can change the text of the edit control.
 SelTextW (see page 103)	Specifies the selected portion of the edit control's text (Unicode version).
 ShowHint (see page 104)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 StatusWidth (see page 104)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 TabOrder (see page 104)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 104)	Determines if the user can tab to a control.
 Text (see page 104)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 104)	Specifies the text string that is displayed in the edit box (Ansi version).
 Visible (see page 104)	Determines whether the component appears on screen.

**TBtEdit Events****TBtEdit Class**

TBtEdit Class	Description
 OnButtonClick (see page 78)	Occurs when the user clicks the button.

**TCustomEditEx Class**

TCustomEditEx Class	Description
 OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.

OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.

**Legend**

	Constructor
	virtual
	protected
	Property
	read only
	Event

**TBtEdit Events****TBtEdit Class**

TBtEdit Class	Description
OnButtonClick (see page 78)	Occurs when the user clicks the button.

**TCustomEditEx Class**

TCustomEditEx Class	Description
OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.

**TUnicodeEdit Methods**










TUnicodeEdit Methods	Description
Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
Destroy (see page 108)	Destroys an instance of TUnicodeEdit.

**TBtEdit Class**





TBtEdit Class	Description
AdjustClientRect (see page 73)	Overrides the inherited method.
ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
Create (see page 73)	Creates and initializes a TBtEdit (see page 70) instance.
CreateParams (see page 73)	Overrides the inherited method.
CreateWnd (see page 73)	Overrides the inherited method.
Destroy (see page 74)	Destroys an instance of TBtEdit (see page 70)
EndTracking (see page 74)	Called after finishing the mouse tracking
KeyDown (see page 74)	Overrides the inherited method.
KeyPress (see page 74)	Overrides the inherited method.
MouseMove (see page 74)	Overrides the inherited method.
MouseUp (see page 75)	Overrides the inherited method.
Paint (see page 75)	Overrides the base rendering method.
PaintBtnGlyph (see page 75)	Renders the image of the button.
PaintStatus (see page 75)	Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.
PaintWindow (see page 75)	Overrides the inherited method.
PtInButton (see page 75)	Checks if specified point is over the button
StartTracking (see page 76)	Calls immediately after user pushes the button.
StopTracking (see page 76)	Calls immediately after user releases the button.
TrackButton (see page 76)	Controls tracking button process.

**TCustomEditEx Class**










TCustomEditEx Class	Description
AcceptListValue (see page 82)	Active pop-up window.
ButtonClick (see page 82)	Simulates a button click, as if the user had clicked the button.
CloseUp (see page 82)	Generates an OnCloseUp (see page 85) event and makes some other actions.
Create (see page 82)	Creates and initializes a TBtEdit instance.

 Destroy (see page 82)	Destroys an instance of TBtnEdit
 DoDropDownKeys (see page 83)	Works as a method dispatcher depending on parameter values.
 DropDown (see page 83)	Generates an OnDropDown (see page 85) event.
 EndTracking (see page 83)	Called after finishing the mouse tracking
 KeyPress (see page 83)	Respond to keyboard input.
 MouseDown (see page 83)	Overrides inherited method
 MouseMove (see page 83)	Overrides the inherited method.
 PaintBtnGlyph (see page 84)	Overrides inherited method
 StartTracking (see page 84)	Overrides inherited property





## TUnicodeEdit Properties

TUnicodeEdit Properties	Description
 IsUnicode (see page 108)	Specifies whether control is Unicode edit.
 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).


















## TBtnEdit Class

TBtnEdit Class	Description
 Alignment (see page 76)	Determines how the text is aligned within the editor control.
 ButtonVisible (see page 76)	Specifies if button-like rectangle at the right edge of the control is visible.
 ButtonWidth (see page 77)	Specifies width in pixels of the button.
 Canvas (see page 77)	Provides access to the drawing surface of the TBtnEdit (see page 70).
 MultiLine (see page 77)	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth (see page 77)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
 WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

















































## TCustomEditEx Class

TCustomEditEx Class	Description
 ActiveList (see page 84)	Specifies current popup control.
 EditStyle (see page 84)	Specifies edit style of the button
 ListAlign (see page 84)	Specifies relative align of popup control.
 PickList (see page 84)	Determines the drop-down list.

## TEditEx Class

TEditEx Class	Description
 Alignment (see page 92)	Determines how the text is aligned within the editor control.
 Anchors (see page 92)	Specifies how the control is anchored to its parent.
 AutoSelect (see page 92)	Determines whether all the text in the edit control is automatically selected when the control gets focus.
 AutoSize (see page 92)	Determines whether the height of the edit control automatically resizes to accommodate the text.
 BevelEdges (see page 93)	Specifies which edges of the control are beveled.
 BevelInner (see page 93)	Specifies the cut of the inner bevel.
 BevelKind (see page 93)	Specifies the control's bevel style.
 BevelOuter (see page 93)	Specifies the cut of the outer bevel.
 BevelWidth (see page 93)	Specifies the width of the inner and outer bevels.
 BiDiMode (see page 94)	Specifies the bi-directional mode for the control.
 BorderStyle (see page 94)	Determines the style of the line drawn around the perimeter of the control.
 ButtonWidth (see page 94)	Specifies width in pixels of the button.
 CharCase (see page 94)	Determines the case of the text within the edit control.
 Color (see page 95)	Specifies the background color of the control.
 Constraints (see page 95)	Specifies the size constraints for the control.
 Ctl3D (see page 95)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DragCursor (see page 95)	Indicates the image used to represent the mouse pointer when the control is being dragged.



 DragKind (see page 95)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 95)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 EditMask (see page 96)	Specifies the mask that represents what text is valid for the masked edit control.
 EditStyle (see page 96)	Specifies edit style of the button
 Enabled (see page 96)	Controls whether the control responds to mouse, keyboard, and timer events.
 Font (see page 96)	Controls the attributes of text written on or in the control.
 ImeMode (see page 96)	Determines the behavior of the input method editor (IME).
 ImeName (see page 97)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
 IsUnicode (see page 97)	Specifies whether control is Unicode edit.
 ListAlign (see page 97)	Specifies relative align of popup control.
 MaxLength (see page 97)	Specifies the maximum number of characters that can appear in the edit control.
 OnAcceptListValue (see page 98)	Occurs when the user makes right choice in the drop-down list.
 OnButtonClick (see page 98)	Occurs when the user clicks the button.
 OnChange (see page 98)	Occurs when the text for the edit control may have changed.
 OnClick (see page 98)	Occurs when the user clicks the control.
 OnCloseUp (see page 98)	Occurs when the drop-down list closes up due to some user action.
 OnDbClick (see page 98)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 98)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 99)	Occurs when the user drags an object over a control.
 OnDropDown (see page 99)	Occurs when the user opens the drop-down list.
 OnEndDrag (see page 99)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 99)	Occurs when a control receives the input focus.
 OnExit (see page 100)	Occurs when the input focus shifts away from one control to another.
 OnKeyDown (see page 100)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 100)	Occurs when key pressed.
 OnKeyUp (see page 100)	Occurs when the user releases a key that has been pressed.
 OnMeasureWidth (see page 101)	Occurs when width of controls needs to be calculated.
 OnMouseDown (see page 101)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 101)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 101)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnStartDrag (see page 102)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 102)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 102)	Determines where a control looks for its color information.
 ParentCtl3D (see page 102)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 102)	Determines where a control looks for its font information.
 ParentShowHint (see page 102)	Determines where a control looks to find out if its Help Hint should be shown.
 PasswordChar (see page 103)	Indicates the character, if any, to display in place of the actual characters typed in the control.
 PickList (see page 103)	Determines the drop-down list.
 PopupMenu (see page 103)	Identifies the pop-up menu associated with the control.
 ReadOnly (see page 103)	Determines whether the user can change the text of the edit control.
 SelTextW (see page 103)	Specifies the selected portion of the edit control's text (Unicode version).
 ShowHint (see page 104)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 StatusWidth (see page 104)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 TabOrder (see page 104)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 104)	Determines if the user can tab to a control.
 Text (see page 104)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 104)	Specifies the text string that is displayed in the edit box (Ansi version).
 Visible (see page 104)	Determines whether the component appears on screen.



### 1.11.1.3.1 TEditEx Properties

#### 1.11.1.3.1.1 TEditEx.Alignment Property

Determines how the text is aligned within the editor control.

```
property Alignment: TAlignment;
```

##### Description

Use Alignment to change the way the text is formatted by the in-place editor control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control
taRightJustify	Align text to the right side of the control

#### 1.11.1.3.1.2 TEditEx.Anchors Property

Specifies how the control is anchored to its parent.

```
property Anchors;
```

##### Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

**Note:** If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

#### 1.11.1.3.1.3 TEditEx.AutoSelect Property

Determines whether all the text in the edit control is automatically selected when the control gets focus.

```
property AutoSelect;
```

##### Description

Set AutoSelect to select all the text when the edit control gets focus. AutoSelect only applies to single-line edit controls.

Use AutoSelect when the user is more likely to replace the text in the edit control than to append to it.

#### 1.11.1.3.1.4 TEditEx.AutoSize Property

Determines whether the height of the edit control automatically resizes to accommodate the text.

```
property AutoSize;
```

#### Description

Use AutoSize to make the edit control adjust its size automatically so the client area accommodates the height of the text. When AutoSize is false, the edit control has a fixed height. When AutoSize is true, the size of the control is readjusted whenever a change occurs that could affect the height of the control, such as a change to the font or border style.

### 1.11.1.3.1.5 TEditEx.BevelEdges Property

Specifies which edges of the control are beveled.

```
property BevelEdges;
```

#### Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

### 1.11.1.3.1.6 TEditEx.BevelInner Property

Specifies the cut of the inner bevel.

```
property BevelInner;
```

#### Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

### 1.11.1.3.1.7 TEditEx.BevelKind Property

Specifies the control's bevel style.

```
property BevelKind;
```

#### Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

### 1.11.1.3.1.8 TEditEx.BevelOuter Property

Specifies the cut of the outer bevel.

```
property BevelOuter;
```

#### Description

Use BevelOuter to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

### 1.11.1.3.1.9 TEditEx.BevelWidth Property

Specifies the width of the inner and outer bevels.

```
property BevelWidth;
```

**Description**

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

**1.11.1.3.1.10 TEditEx.BiDiMode Property**

Specifies the bi-directional mode for the control.

```
property BiDiMode;
```

**Description**

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

**1.11.1.3.1.11 TEditEx.BorderStyle Property**

Determines the style of the line drawn around the perimeter of the control.

```
property BorderStyle;
```

**Description**

Use BorderStyle to specify whether the control has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

**1.11.1.3.1.12 TEditEx.ButtonWidth Property**

Specifies width in pixels of the button.

```
property ButtonWidth: integer;
```

**Description**

This property specifies width in pixels of the button if any.

Use this property to read or set width of the button.

**1.11.1.3.1.13 TEditEx.CharCase Property**

Determines the case of the text within the edit control.

```
property CharCase;
```

**Description**

Use CharCase to force the contents of the edit control to assume a particular case.

When CharCase is set to ecLowerCase or ecUpperCase, the case of characters is converted as the user types them into the edit control. Changing the CharCase property to ecLowerCase or ecUpperCase changes the actual contents of the text, not just the appearance. Any case information is lost and can't be recaptured by changing CharCase to ecNormal.

#### 1.11.1.3.1.14 TEditEx.Color Property

Specifies the background color of the control.

```
property Color;
```

##### Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

#### 1.11.1.3.1.15 TEditEx.Constraints Property

Specifies the size constraints for the control.

```
property Constraints;
```

##### Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

**Warning:** Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

#### 1.11.1.3.1.16 TEditEx.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

```
property Ctl3D;
```

##### Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

#### 1.11.1.3.1.17 TEditEx.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

```
property DragCursor;
```

##### Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

**Note:** To make a custom cursor available for the DragCursor property, see the Cursor property.

#### 1.11.1.3.1.18 TEditEx.DragKind Property

Specifies whether the control is being dragged normally or for docking.

```
property DragKind;
```

##### Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

#### 1.11.1.3.1.19 TEditEx.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

```
property DragMode;
```

#### Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

### 1.11.1.3.1.20 TEditEx.EditMask Property

Specifies the mask that represents what text is valid for the masked edit control.

```
property EditMask;
```

#### Description

Use EditMask to restrict the characters a user can enter into the masked edit control to valid characters and formats. If the user attempts to enter an invalid character, the edit control does not accept the character. Validation is performed on a character-by-character basis by the ValidateEdit method.

Setting EditMask to an empty string removes the mask.

### 1.11.1.3.1.21 TEditEx.EditStyle Property

Specifies edit style of the button

```
property EditStyle: TInplaceEditStyle;
```

#### Description

Specifies of what visual representation the button will be.

There are three styles available

- ieSimple - without button
- ieEllipsis - button with ellipsis
- iePickList - button with drop-down list

### 1.11.1.3.1.22 TEditEx.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

```
property Enabled;
```

#### Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

### 1.11.1.3.1.23 TEditEx.Font Property

Controls the attributes of text written on or in the control.

```
property Font;
```

#### Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

### 1.11.1.3.1.24 TEditEx.ImeMode Property

Determines the behavior of the input method editor (IME).

```
property ImeMode;
```

#### Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

### 1.11.1.3.1.25 TEditEx.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

```
property ImeName;
```

#### Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

### 1.11.1.3.1.26 TEditEx.IsUnicode Property

Specifies whether control is Unicode edit.

```
property IsUnicode: Boolean;
```

#### Description

Set IsUnicode to True to make edit control Unicode window. When control is Unicode TextW (see page 109) and SelTextW (see page 109) properties should be used instead of Text and SelText properties.

### 1.11.1.3.1.27 TEditEx.ListAlign Property

Specifies relative align of popup control.

```
property ListAlign: TAlignment;
```

### 1.11.1.3.1.28 TEditEx.MaxLength Property

Specifies the maximum number of characters that can appear in the edit control.

```
property MaxLength;
```

#### Description

MaxLength is the length of the EditText. Set MaxLength to limit the number of characters that can appear in the edit control when there is no EditMask. If there is an EditMask, MaxLength is implied by the mask itself, and cannot be changed. The EditText contains blank characters for each character that has not been entered, padding the end or beginning of every optional section, so it remains constant in length.

### 1.11.1.3.1.29 TEditEx.OnAcceptListValue Property

Occurs when the user makes right choice in the drop-down list.

**property** OnAcceptListValue: TOnAcceptListValueEvent;

#### Description

Write an OnAcceptListValue event handler to implement special processing that needs to occur when the user makes a choice in the drop-down list. For example, an OnAcceptListValue event handler can check whether the user can make or not such choice.

### 1.11.1.3.1.30 TEditEx.OnButtonClick Property

Occurs when the user clicks the button.

**property** OnButtonClick: TNotifyEvent;

#### Description

Use the OnButtonClick event handler to respond when the user clicks the button or do some equivalent action such as pressing the associated short keys or some. This is very similar to standard OnClick event.

### 1.11.1.3.1.31 TEditEx.OnChange Property

Occurs when the text for the edit control may have changed.

**property** OnChange;

#### Description

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

### 1.11.1.3.1.32 TEditEx.OnClick Property

Occurs when the user clicks the control.

**property** OnClick;

### 1.11.1.3.1.33 TEditEx.OnCloseUp Property

Occurs when the drop-down list closes up due to some user action.

**property** OnCloseUp: TCloseUpEvent;

#### Description

Write an OnCloseUp event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an OnCloseUp event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

### 1.11.1.3.1.34 TEditEx.OnDbClick Property

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

**property** OnDbClick;

#### Description

Use the OnDbClick event to respond to mouse double-clicks.

### 1.11.1.3.1.35 TEditEx.OnDragDrop Property

Occurs when the user drops an object being dragged.

```
property OnDragDrop;
```

#### Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

### 1.11.1.3.1.36 TEditEx.OnDragOver Property

Occurs when the user drags an object over a control.

```
property OnDragOver;
```

#### Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

**Note:** Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

### 1.11.1.3.1.37 TEditEx.OnDropDown Property

Occurs when the user opens the drop-down list.

```
property OnDropDown: TNotifyEvent;
```

#### Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

### 1.11.1.3.1.38 TEditEx.OnEndDrag Property

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

```
property OnEndDrag;
```

#### Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

### 1.11.1.3.1.39 TEditEx.OnEnter Property

Occurs when a control receives the input focus.

```
property OnEnter;
```

#### Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.



The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

#### 1.11.1.3.1.40 TEditEx.OnExit Property

Occurs when the input focus shifts away from one control to another.

```
property OnExit;
```

##### Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

#### 1.11.1.3.1.41 TEditEx.OnKeyDown Property

Occurs when a user presses any key while the control has focus.

```
property OnKeyDown;
```

##### Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys, and pressed mouse buttons.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

#### 1.11.1.3.1.42 TEditEx.OnKeyPress Property

Occurs when key pressed.

```
property OnKeyPress;
```

##### Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

#### 1.11.1.3.1.43 TEditEx.OnKeyUp Property

Occurs when the user releases a key that has been pressed.

```
property OnKeyUp;
```

##### Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

#### 1.11.1.3.1.44 TEditEx.OnMeasureWidth Property

Occurs when width of controls needs to be calculated.

```
property OnMeasureWidth: TMeasureWidthEvent;
```

##### Description

Parameter AWidth represents calculated width, in pixels, of specified in Value text for specified Canvas.

Write an OnMeasureWidth event handler to specify the new user width.

#### 1.11.1.3.1.45 TEditEx.OnMouseDown Property

Occurs when the user presses a mouse button with the mouse pointer over a control.

```
property OnMouseDown;
```

##### Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

#### 1.11.1.3.1.46 TEditEx.OnMouseMove Property

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

```
property OnMouseMove;
```

##### Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

#### 1.11.1.3.1.47 TEditEx.OnMouseUp Property

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

```
property OnMouseUp;
```

##### Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

### 1.11.1.3.1.48 TEditEx.OnStartDrag Property

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

**property** OnStartDrag;

#### Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itse

### 1.11.1.3.1.49 TEditEx.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

**property** ParentBiDiMode;

### 1.11.1.3.1.50 TEditEx.ParentColor Property

Determines where a control looks for its color information.

**property** ParentColor;

### 1.11.1.3.1.51 TEditEx.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

**property** ParentCtl3D;

#### Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

### 1.11.1.3.1.52 TEditEx.ParentFont Property

Determines where a control looks for its font information.

**property** ParentFont;

### 1.11.1.3.1.53 TEditEx.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

**property** ParentShowHint;

### 1.11.1.3.1.54 TEditEx.PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

**property** PasswordChar;

#### Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

### 1.11.1.3.1.55 TEditEx.PickList Property

Determines the drop-down list.

**property** PickList: TPopupListbox;

#### Description

Use this property to access drop-down list object for iePickList EditStyle.

### 1.11.1.3.1.56 TEditEx.PopupMenu Property

Identifies the pop-up menu associated with the control.

**property** PopupMenu;

#### Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

### 1.11.1.3.1.57 TEditEx.ReadOnly Property

Determines whether the user can change the text of the edit control.

**property** ReadOnly;

#### Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

Setting ReadOnly to true ensures that the text is not altered, while still allowing the user to select text. The selected text can then be manipulated by the application, or copied to the Clipboard.

### 1.11.1.3.1.58 TEditEx SelTextW Property

Specifies the selected portion of the edit control's text (Unicode version).

**property** SelTextW: WideString;

#### Description

Read SelTextW to determine the value of the selected text. Set SelTextW to replace the selected text with a new string. If there is no selection, but the edit control has focus, set SelTextW to insert a new string into the text at the cursor.

Use this property when IsUnicode is True. Otherwise this property may corrupt string due to Ansi to Unicode conversion.

### 1.11.1.3.1.59 TEditEx.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

### 1.11.1.3.1.60 TEditEx.StatusWidth Property

Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.

```
property StatusWidth: integer;
```

### 1.11.1.3.1.61 TEditEx.TabOrder Property

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

### 1.11.1.3.1.62 TEditEx.TabStop Property

Determines if the user can tab to a control.

```
property TabStop;
```

#### Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

### 1.11.1.3.1.63 TEditEx.Text Property

Specifies the text string that is displayed in the edit box (Ansi version).

```
property Text;
```

#### Description

Use the Text property to read the text of the edit box or specify a new string for the Text value. By default, Text is the string specified in the Name property.

Use this property when IsUnicode is False. Otherwise this property may corrupt string due to Unicode to Ansi conversion.

### 1.11.1.3.1.64 TEditEx.TextW Property

Specifies the text string that is displayed in the edit box (Ansi version).

```
property TextW: WideString;
```

#### Description

Use the TextW property to read the text of the edit box or specify a new string for the TextW value. By default, TextW is the string specified in the Name property.

Use this property when IsUnicode is True. Otherwise this property may corrupt string due to Ansi to Unicode conversion.

### 1.11.1.3.1.65 TEditEx.Visible Property

Determines whether the component appears on screen.

```
property Visible;
```

## Description

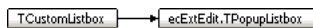
Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

## 1.11.1.4 TPopupListbox Class

TPopupListbox represents drop-down list in TExtEdit type.

### Class Hierarchy



```
TPopupListbox = class(TCustomListbox);
```

### File







ecExtEdit

### Description






Add a description here...

### Members





#### TPopupListbox Methods

TPopupListbox Methods	Description
  CreateParams (see page 106)	Overrides inherited procedure.
  CreateWnd (see page 106)	Overrides inherited procedure.
  KeyPress (see page 106)	Overrides inherited procedure.







#### TPopupListbox Properties

TPopupListbox Properties	Description
 ItemHeight (see page 106)	This is inherited property from TCustomListBox.ItemHeight
 OnDrawItem (see page 106)	This is inherited property from TCustomListBox.OnDrawItem
 OnMeasureItem (see page 107)	This is inherited property from TCustomListBox.OnMeasureItem
 Sorted (see page 107)	This is inherited property from TCustomListBox.Sorted
 Style (see page 107)	This is inherited property from TCustomListBox.Style






### Legend

	Method
	protected
	virtual
	Property

#### TPopupListbox Methods

TPopupListbox Methods	Description
  CreateParams (see page 106)	Overrides inherited procedure.
  CreateWnd (see page 106)	Overrides inherited procedure.
  KeyPress (see page 106)	Overrides inherited procedure.

#### TPopupListbox Properties

TPopupListbox Properties	Description
 ItemHeight (see page 106)	This is inherited property from TCustomListBox.ItemHeight
 OnDrawItem (see page 106)	This is inherited property from TCustomListBox.OnDrawItem
 OnMeasureItem (see page 107)	This is inherited property from TCustomListBox.OnMeasureItem
 Sorted (see page 107)	This is inherited property from TCustomListBox.Sorted
 Style (see page 107)	This is inherited property from TCustomListBox.Style

### 1.11.1.4.1 TPopupListbox Methods

#### 1.11.1.4.1.1 TPopupListbox.CreateParams Method

Overrides inherited procedure.

```
procedure CreateParams(var Params: TCreateParams); override;
```

##### Description

CreateParams

- Calls inherited CreateParams
- Adds WS\_BORDER to Style (see page 107) property
- Adds WS\_EX\_TOOLWINDOW and WS\_EX\_TOPMOST to ExStyle property
- Sets control's window style according to its bi-directional support (via AddBiDiModeExStyle(ExStyle))
- Adds CS\_SAVEBITS flag to WindowClass.Style (see page 107) property

#### 1.11.1.4.1.2 TPopupListbox.CreateWnd Method

Overrides inherited procedure.

```
procedure CreateWnd; override;
```

##### Description

CreateWnd

- Calls inherited CreateWnd
- Correctly sets parent via Windows.SetParent
- Sets focus on a window

#### 1.11.1.4.1.3 TPopupListbox.KeyPress Method

Overrides inherited procedure.

```
procedure KeyPress(var Key: Char); override;
```

##### Description

KeyPress

- searching the items for the first item that matches the string entered so far
- calls inherited

### 1.11.1.4.2 TPopupListbox Properties

#### 1.11.1.4.2.1 TPopupListbox.ItemHeight Property

This is inherited property from TCustomListBox.ItemHeight

```
property ItemHeight;
```

##### Description

Specifies the height in pixels of an item in an owner-draw list box.

See TCustomListBox.ItemHeight for more information

#### 1.11.1.4.2.2 TPopupListbox.OnDrawItem Property

This is inherited property from TCustomListBox.OnDrawItem

**property** OnDrawItem;

**Description**

Occurs when an item in an owner-draw list box needs to be redisplayed.  
See TCustomListBox.OnDrawItem for more information

1.11.1.4.2.3 TPopupListbox.OnMeasureItem Property

This is inherited property from TCustomListBox.OnMeasureItem

**property** OnMeasureItem;

**Description**

Occurs when the application needs to redisplay an item in a variable height owner-draw list box.  
See TCustomListBox.OnMeasureItem for more information

1.11.1.4.2.4 TPopupListbox.Sorted Property

This is inherited property from TCustomListBox.Sorted

**property** Sorted;

**Description**

Specifies whether the items in a list box are arranged alphabetically.  
See TCustomListBox.Sorted for more information

1.11.1.4.2.5 TPopupListbox.Style Property

This is inherited property from TCustomListBox.Style

**property** Style;

**Description**

Determines whether the list box is standard or owner-draw and whether it is virtual.  
See TCustomListBox.Style for more information

1.11.1.5 TUnicodeEdit Class

Mask edit control with Unicode support.

**Class Hierarchy**



TUnicodeEdit = **class**(TCustomMaskEdit);

**File**

ecExtEdit

**Description**

TUnicodeEdit may be used as ANSI or Unicode edit control. Type of control is defined by the IsUnicode (see page 108) property. When control is Unicode TextW (see page 109) and SelTextW (see page 109) properties should be used instead of Text (see page 109) and SelText properties.

**Members**





**TUnicodeEdit Methods**

TUnicodeEdit Methods	Description
Create (see page 108)	Creates and initializes a TUnicodeEdit instance.







 Destroy (see page 108)	Destroys an instance of TUnicodeEdit.
----------------------------------------------------------------------------------------------------------	---------------------------------------


## TUnicodeEdit Properties

TUnicodeEdit Properties	Description
 IsUnicode (see page 108)	Specifies whether control is Unicode edit.
 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).





## Legend

	Constructor
	virtual
	Property
	protected

## TUnicodeEdit Methods

TUnicodeEdit Methods	Description
 Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
 Destroy (see page 108)	Destroys an instance of TUnicodeEdit.

## TUnicodeEdit Properties

TUnicodeEdit Properties	Description
 IsUnicode (see page 108)	Specifies whether control is Unicode edit.
 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).

## 1.11.1.5.1 TUnicodeEdit Methods

### 1.11.1.5.1.1 TUnicodeEdit.Create Constructor

Creates and initializes a TUnicodeEdit instance.

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

#### Description

Use Create to programmatically instantiate a TUnicodeEdit component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

### 1.11.1.5.1.2 TUnicodeEdit.Destroy Destructor

Destroys an instance of TUnicodeEdit.

```
destructor Destroy; override;
```

#### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

## 1.11.1.5.2 TUnicodeEdit Properties

### 1.11.1.5.2.1 TUnicodeEdit.IsUnicode Property

Specifies whether control is Unicode edit.

```
property IsUnicode: Boolean;
```

Description

Set IsUnicode to True to make edit control Unicode window. When control is Unicode TextW (see page 109) and SelTextW (see page 109) properties should be used instead of Text (see page 109) and SelText properties.

1.11.1.5.2.2 TUnicodeEdit.SelTextW Property

Specifies the selected portion of the edit control's text (Unicode version).

property SelTextW: WideString;

Description

Read SelTextW to determine the value of the selected text. Set SelTextW to replace the selected text with a new string. If there is no selection, but the edit control has focus, set SelTextW to insert a new string into the text at the cursor.

Use this property when IsUnicode (see page 108) is True. Otherwise this property may corrupt string due to Ansi to Unicode conversion.

1.11.1.5.2.3 TUnicodeEdit.Text Property

Specifies the text string that is displayed in the edit box (Ansi version).

property Text;

Description

Use the Text property to read the text of the edit box or specify a new string for the Text value. By default, Text is the string specified in the Name property.

Use this property when IsUnicode (see page 108) is False. Otherwise this property may corrupt string due to Unicode to Ansi conversion.

1.11.1.5.2.4 TUnicodeEdit.TextW Property

Specifies the text string that is displayed in the edit box (Ansi version).

property TextW: WideString;

Description

Use the TextW property to read the text of the edit box or specify a new string for the TextW value. By default, TextW is the string specified in the Name property.

Use this property when IsUnicode (see page 108) is True. Otherwise this property may corrupt string due to Ansi to Unicode conversion.

1.11.2 Structs, Records, Enums

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

Enumerations

Enumeration	Description
 TInplaceEditStyle (see page 110)	Style of the editor

Legend

	Enumeration
-------------------------------------------------------------------------------------	-------------

### 1.11.2.1 ecExtEdit.TInplaceEditStyle Enumeration

```
TInplaceEditStyle = (  
    ieSimple,  
    ieEllipsis,  
    iePickList  
);
```

**File**  
ecExtEdit

**Members**

Members	Description
ieSimple	Editor without button.
ieEllipsis	Editor with button
iePickList	Editor with drop-down button.

**Description**  
Style of the editor

## 1.11.3 Types

The following table lists types in this documentation.

**Types**

Type	Description
TCloseUpEvent (see page 110)	See TExtEdit.OnCloseUp Event (see page 98)
TMeasureWidthEvent (see page 110)	See TExtEdit.OnMeasureWidth Event (see page 101)
TOnAcceptListValueEvent (see page 111)	See TExtEdit.OnAcceptListValue Event (see page 98)

### 1.11.3.1 ecExtEdit.TCloseUpEvent Type

```
TCloseUpEvent = procedure (Sender: TObject; var Accept: Boolean) of object;
```

**File**  
ecExtEdit

**Description**  
See TExtEdit.OnCloseUp Event (see page 98)

### 1.11.3.2 ecExtEdit.TMeasureWidthEvent Type

```
TMeasureWidthEvent = procedure (Sender: TObject; const Value: string; Canvas: TCanvas; var  
AWidth: Integer) of object;
```

**File**  
ecExtEdit

**Description**  
See TExtEdit.OnMeasureWidth Event (see page 101)

### 1.11.3.3 ecExtEdit.TOnAcceptListValueEvent Type

TOnAcceptListValueEvent = **procedure** (Sender: TObject; **var** ListValue: **string**) **of object**;

File

ecExtEdit

Description

See TTextEdit.OnAcceptListValue Event (🔗 see page 98)

## 1.12 ecHintHelper Namespace

### 1.12.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TecHintHelper (🔗 see page 111)	Automates hints processing and provides properties to adjust it.

#### 1.12.1.1 TecHintHelper Class

Automates hints processing and provides properties to adjust it.

Class Hierarchy



TecHintHelper = **class**(TPersistent);

File

ecHintHelper

Description

Provides properties for hints processing. Supports Unicode hints associated with regions of the control.








To use hint helper you need to call ControlWndProc (🔗 see page 112) from window procedure of control and process CM\_GETHINTDATA (🔗 see page 116) message.

Members




TecHintHelper Methods

TecHintHelper Methods	Description
🔗 CancelHint (🔗 see page 112)	Cancels the display of a hint for a control.
🔗 ControlWndProc (🔗 see page 112)	Owner control passes window messages to this procedure.
🔗 Create (🔗 see page 112)	Creates and initializes a TecHintHelper instance.
🔗 Destroy (🔗 see page 113)	Destroys an instance of TecHintHelper.
🔗 ResetHint (🔗 see page 113)	Hides the current hint.
🔗 ShowHint (🔗 see page 113)	Shows hint window if mouse over region of control which has hint information. Immediately parameter forces to show hint immediately.







## TechHintHelper Properties

TechHintHelper Properties	Description
 CanMoveLeft ( <a href="#">see page 113</a> )	Specifies whether hint may be moved left if it's right side out of screen.
 Color ( <a href="#">see page 113</a> )	Determines the color of the hint boxes for the Help Hints.
 Enabled ( <a href="#">see page 113</a> )	Specifies whether hints should be shown.
 Font ( <a href="#">see page 113</a> )	Specifies font of the hint window.
 HidePause ( <a href="#">see page 114</a> )	Specifies the time interval to wait before hiding the Hint if the mouse has not moved from the control region.
 Pause ( <a href="#">see page 114</a> )	Specifies the time interval that passes before the control's Hint appears when the user places the mouse pointer on a control.
 ShortPause ( <a href="#">see page 114</a> )	Specifies the time period to wait before bringing up a hint if another hint has already been shown for this control.








## Legend

	Method
	virtual
	Property

## TechHintHelper Methods

TechHintHelper Methods	Description
 CancelHint ( <a href="#">see page 112</a> )	Cancels the display of a hint for a control.
 ControlWndProc ( <a href="#">see page 112</a> )	Owner control passes window messages to this procedure.
 Create ( <a href="#">see page 112</a> )	Creates and initializes a TechHintHelper instance.
 Destroy ( <a href="#">see page 113</a> )	Destroys an instance of TechHintHelper.
 ResetHint ( <a href="#">see page 113</a> )	Hides the current hint.
 ShowHint ( <a href="#">see page 113</a> )	Shows hint window if mouse over region of control which has hint information. Immediately parameter forces to show hint immediately.

## TechHintHelper Properties

TechHintHelper Properties	Description
 CanMoveLeft ( <a href="#">see page 113</a> )	Specifies whether hint may be moved left if it's right side out of screen.
 Color ( <a href="#">see page 113</a> )	Determines the color of the hint boxes for the Help Hints.
 Enabled ( <a href="#">see page 113</a> )	Specifies whether hints should be shown.
 Font ( <a href="#">see page 113</a> )	Specifies font of the hint window.
 HidePause ( <a href="#">see page 114</a> )	Specifies the time interval to wait before hiding the Hint if the mouse has not moved from the control region.
 Pause ( <a href="#">see page 114</a> )	Specifies the time interval that passes before the control's Hint appears when the user places the mouse pointer on a control.
 ShortPause ( <a href="#">see page 114</a> )	Specifies the time period to wait before bringing up a hint if another hint has already been shown for this control.

## 1.12.1.1.1 TechHintHelper Methods

### 1.12.1.1.1.1 TechHintHelper.CancelHint Method

Cancels the display of a hint for a control.

```
procedure CancelHint;
```

### 1.12.1.1.1.2 TechHintHelper.ControlWndProc Method

Owner control passes window messages to this procedure.

```
procedure ControlWndProc(var Message: TMessage); virtual;
```

### 1.12.1.1.1.3 TechHintHelper.Create Constructor

Creates and initializes a TechHintHelper instance.

```
constructor Create(AOwner: TControl);
```

**Description**

Use Create to programmatically instantiate a TechHintHelper object.

AOwner specifies control which owns hint helper.

**1.12.1.1.1.4 TechHintHelper.Destroy Destructor**

Destroys an instance of TechHintHelper.

```
destructor Destroy; override;
```

**Description**

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

**1.12.1.1.1.5 TechHintHelper.ResetHint Method**

Hides the current hint.

```
procedure ResetHint;
```

**1.12.1.1.1.6 TechHintHelper.ShowHint Method**

Shows hint window if mouse over region of control which has hint information. Immediately parameter forces to show hint immediately.

```
procedure ShowHint(Immediately: Boolean);
```

**1.12.1.1.2 TechHintHelper Properties****1.12.1.1.2.1 TechHintHelper.CanMoveLeft Property**

Specifies whether hint may be moved left if it's right side out of screen.

```
property CanMoveLeft: Boolean;
```

**1.12.1.1.2.2 TechHintHelper.Color Property**

Determines the color of the hint boxes for the Help Hints.

```
property Color: TColor;
```

**Description**

Use Color to specify the hint box color. A default color value of clInfoBk is set for the Color property in the constructor when the hint helper is created.

**1.12.1.1.2.3 TechHintHelper.Enabled Property**

Specifies whether hints should be shown.

```
property Enabled: Boolean;
```

**Description**

Hint processing provided by the TechHintHelper (see page 111) differs from standard hint processing. ShowHint (see page 113) property of control is used only for standard hint processing and is not used in TechHintHelper (see page 111). To enable/ disable hints for control use TechHintHelper.Enabled property.

**1.12.1.1.2.4 TechHintHelper.Font Property**

Specifies font of the hint window.

```
property Font: TFont;
```

### 1.12.1.1.2.5 TecHintHelper.HidePause Property

Specifies the time interval to wait before hiding the Hint if the mouse has not moved from the control region.

```
property HidePause: Integer;
```

**Description**

Use HidePause to specify a wait time in milliseconds, which is different from the default value of 2500 ms or 2 1/2 seconds that is set in the constructor.

### 1.12.1.1.2.6 TecHintHelper.Pause Property

Specifies the time interval that passes before the control's Hint appears when the user places the mouse pointer on a control.

```
property Pause: Integer;
```

**Description**

Use Pause to change the default pause time of 500 ms or 1/2 second that is set in the constructor. When assigning a value to Pause, specify the interval in milliseconds.

### 1.12.1.1.2.7 TecHintHelper.ShortPause Property

Specifies the time period to wait before bringing up a hint if another hint has already been shown for this control.

```
property ShortPause: Integer;
```



**Description**

Use ShortPause to reduce the flicker when moving the mouse quickly over, for example, a set of buttons that all have help hints on. Specify a value in milliseconds that is different from the default value of 50 ms that is set in the constructor.

## 1.12.2 Structs, Records, Enums

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

### Records

Record	Description
 TCMGetHintData (see page 114)	Message record passed to control as CM_GETHINTDATA (see page 116) message.
 TecHintData (see page 115)	Hint information structure filled by the control.

### Legend

	Record
-------------------------------------------------------------------------------------	--------

### 1.12.2.1 ecHintHelper.TCMGetHintData Record

Message record passed to control as CM\_GETHINTDATA (see page 116) message.

```
TCMGetHintData = packed record
  Msg: Cardinal;
  Pt: TSmallPoint;
  pHintData: PecHintData;
  Result: Longint;
end;
```

**File**

ecHintHelper

Members

Members	Description
Msg: Cardinal;	Message code which is equal to CM_GETHINTDATA (see page 116).
Pt: TSmallPoint;	Mouse position in client coordinates.
pHintData: PecHintData;	Pointer to hint information structure to be filled by the control.
Result: Longint;	Result value - not used.

1.12.2.2 ecHintHelper.TecHintData Record

Hint information structure filled by the control.

```
TecHintData = record
  Text: WideString;
  HintSense: TRect;
  BaseRect: TRect;
end;
```

File

ecHintHelper

Members

Members	Description
Text: WideString;	Unicode hint text.
HintSense: TRect;	Rectangle in which caret movements does not effect hint showing, for example, item rectangle.
BaseRect: TRect;	Rectangle around which hint window will be displayed.

1.12.3 Types

The following table lists types in this documentation.

Types

Type	Description
PecHintData (see page 115)	Pointer to TecHintData (see page 115).

1.12.3.1 ecHintHelper.PecHintData Type

Pointer to TecHintData (see page 115).

```
PecHintData = ^TecHintData;
```

File

ecHintHelper

1.12.4 Constants

The following table lists constants in this documentation.

Constants

Constant	Description
CM_GETHINTDATA (see page 116)	Message which should be processed by control to define hint information.



### 1.12.4.1 ecHintHelper.CM\_GETHINTDATA Constant

Message which should be processed by control to define hint information.

```
CM_GETHINTDATA = $7B00;
```

**File**

ecHintHelper

## 1.13 ecToolList Namespace

### 1.13.1 Classes

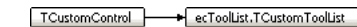
The following table lists classes in this documentation.

Class	Description
TCustomToolList (🔗 see page 116)	Tool list represents item list organized in categories.
TToolItemStyle (🔗 see page 123)	Holds visual properties of tool item.
TToolList (🔗 see page 125)	Tool list represents item list organized in categories.
TToolListItem (🔗 see page 141)	Represents tool item used in tool lists (🔗 see page 116).
TToolListItems (🔗 see page 143)	Collection of tool list items.

#### 1.13.1.1 TCustomToolList Class

Tool list represents item list organized in categories.

**Class Hierarchy**



```
TCustomToolList = class(TCustomControl);
```

**File**

ecToolList














**Description**

Items (🔗 see page 121) in tool list has caption, image and hint. Categories may be collapsed. Items (🔗 see page 121) may be filtered.

























**Members**

**TCustomToolList Methods**


TCustomToolList Methods	Description
🔗 CollapseAll (🔗 see page 118)	Collapses all categories.
🔗 Create (🔗 see page 119)	Creates and initializes a TCustomToolList instance.
🔗 Destroy (🔗 see page 119)	Destroys an instance of TCustomToolList.
🔗 DrawItemImage (🔗 see page 119)	Draws item image.
🔗 ExpandAll (🔗 see page 119)	Expands all categories.
🔗 GetCategoryItem (🔗 see page 119)	Returns index of category item at the given position or above.
🔗 ItemAtPos (🔗 see page 119)	Returns items at the given position. If there are no item at the specified position function returns -1.

  ItemIndexChanged (see page 119)	Called when selected item was changed.
 ItemRect (see page 119)	Returns rectangle occupied by the item.
  ItemsArranged (see page 120)	Called after items were rearranged by the drag&drop operations.
  ItemsChanged (see page 120)	Called when items were changed (any changes).
 ItemsHeight (see page 120)	Calculates total height of items.
 MakeTopItem (see page 120)	Scrolls list to make specified item topmost item.
 MakeVisible (see page 120)	Scrolls list to make specified item visible.
  PaintItem (see page 120)	Calls TToolListItem.Paint and allows to customize item rendering in derived classes.
 SelectFirstVisible (see page 120)	Selects first visible, i.e. not hidden, item.







### TCustomToolList Properties

TCustomToolList Properties	Description
 AllowArrange (see page 120)	Specifies whether items can be arranged by the drag&drop operations.
 AutoCollapse (see page 120)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
 CategoryHeight (see page 120)	Specifies height of category item.
 Filtered (see page 120)	Specifies whether items are filtered.
 FilterString (see page 121)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 121)	Holds folding icon images.
 HintProps (see page 121)	Provide properties to adjust hints processing.
 Images (see page 121)	Determines which image list is associated with the tool list.
 InsertAtItem (see page 121)	Specifies item index at which dragged object can be dropped.
 ItemHeight (see page 121)	Specifies height of the normal item.
 ItemIndex (see page 121)	Indicates which item is selected. If no item is selected ItemIndex is equal to -1.
 Items (see page 121)	Provides access to items displayed in tool list.
 MouseOverItem (see page 122)	Indicates item over which mouse cursor is located.
 RightClickSelect (see page 122)	Specifies whether item can be selected by mouse right click.
 RowSpace (see page 122)	Specifies space between two sequential items.
  Selected (see page 122)	Currently selected item in tool list.
 StyleCategory (see page 122)	Specifies style of category items.
 StyleCategoryMouseOver (see page 122)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (see page 122)	Specifies style of selected category item.
 StyleItem (see page 122)	Specifies style of tool items.
 StyleItemMouseOver (see page 122)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (see page 122)	Specifies style of selected tool item.
 VerticalGroups (see page 122)	Specifies whether category item should be displayed vertically along owned items.
 ViewOrigin (see page 123)	Specifies scrolling position of the tool list control.

### TCustomToolList Events

TCustomToolList Events	Description
 OnItemArranged (see page 123)	Occurs when items order was changed.
 OnItemChanged (see page 123)	Occurs when selected item is changes.

















### Legend

	Method
	virtual
	protected
	Property
	read only
	Event






### TCustomToolList Events

TCustomToolList Events	Description
 OnItemArranged (see page 123)	Occurs when items order was changed.
 OnItemChanged (see page 123)	Occurs when selected item is changes.

**TCustomToolList Methods**

<b>TCustomToolList Methods</b>	<b>Description</b>
 CollapseAll (see page 118)	Collapses all categories.
 Create (see page 119)	Creates and initializes a TCustomToolList instance.
 Destroy (see page 119)	Destroys an instance of TCustomToolList.
 DrawItemImage (see page 119)	Draws item image.
 ExpandAll (see page 119)	Expands all categories.
 GetCategoryItem (see page 119)	Returns index of category item at the given position or above.
 ItemAtPos (see page 119)	Returns items at the given position. If there are no item at the specified position function returns -1.
 ItemIndexChanged (see page 119)	Called when selected item was changed.
 ItemRect (see page 119)	Returns rectangle occupied by the item.
 ItemsArranged (see page 120)	Called after items were rearranged by the drag&drop operations.
 ItemsChanged (see page 120)	Called when items were changed (any changes).
 ItemsHeight (see page 120)	Calculates total height of items.
 MakeTopItem (see page 120)	Scrolls list to make specified item topmost item.
 MakeVisible (see page 120)	Scrolls list to make specified item visible.
 PaintItem (see page 120)	Calls TToolListItem.Paint and allows to customize item rendering in derived classes.
 SelectFirstVisible (see page 120)	Selects first visible, i.e. not hidden, item.

**TCustomToolList Properties**

<b>TCustomToolList Properties</b>	<b>Description</b>
 AllowArrange (see page 120)	Specifies whether items can be arranged by the drag&drop operations.
 AutoCollapse (see page 120)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
 CategoryHeight (see page 120)	Specifies height of category item.
 Filtered (see page 120)	Specifies whether items are filtered.
 FilterString (see page 121)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 121)	Holds folding icon images.
 HintProps (see page 121)	Provide properties to adjust hints processing.
 Images (see page 121)	Determines which image list is associated with the tool list.
 InsertAtItem (see page 121)	Specifies item index at which dragged object can be dropped.
 ItemHeight (see page 121)	Specifies height of the normal item.
 ItemIndex (see page 121)	Indicates which item is selected. If no item is selected ItemIndex is equal to -1.
 Items (see page 121)	Provides access to items displayed in tool list.
 MouseOverItem (see page 122)	Indicates item over which mouse cursor is located.
 RightClickSelect (see page 122)	Specifies whether item can be selected by mouse right click.
 RowSpace (see page 122)	Specifies space between two sequential items.
 Selected (see page 122)	Currently selected item in tool list.
 StyleCategory (see page 122)	Specifies style of category items.
 StyleCategoryMouseOver (see page 122)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (see page 122)	Specifies style of selected category item.
 StyleItem (see page 122)	Specifies style of tool items.
 StyleItemMouseOver (see page 122)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (see page 122)	Specifies style of selected tool item.
 VerticalGroups (see page 122)	Specifies whether category item should be displayed vertically along owned items.
 ViewOrigin (see page 123)	Specifies scrolling position of the tool list control.

**1.13.1.1.1 TCustomToolList Methods****1.13.1.1.1.1 TCustomToolList.CollapseAll Method**

Collapses all categories.

**procedure** CollapseAll;

### 1.13.1.1.1.2 TCustomToolList.Create Constructor

Creates and initializes a TCustomToolList instance.

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

#### Description

Use Create to programmatically instantiate a TCustomToolList component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

### 1.13.1.1.1.3 TCustomToolList.Destroy Destructor

Destroys an instance of TCustomToolList.

```
destructor Destroy; override;
```

#### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

### 1.13.1.1.1.4 TCustomToolList.DrawItemImage Method

Draws item image.

```
procedure DrawItemImage(Item: TToolListItem; Canvas: TCanvas; var R: TRect; State: TToolItemState); virtual;
```

#### Description

By default, it draws image from image list of the tool list, but in derived classes it maybe redefined to get images from another source.

### 1.13.1.1.1.5 TCustomToolList.ExpandAll Method

Expands all categories.

```
procedure ExpandAll;
```

### 1.13.1.1.1.6 TCustomToolList.GetCategoryItem Method

Returns index of category item at the given position or above.

```
function GetCategoryItem(Index: integer): integer;
```

### 1.13.1.1.1.7 TCustomToolList.ItemAtPos Method

Returns items at the given position. If there are no item at the specified position function returns -1.

```
function ItemAtPos(const p: TPoint): integer;
```

### 1.13.1.1.1.8 TCustomToolList.ItemIndexChanged Method

Called when selected item was changed.

```
procedure ItemIndexChanged; virtual;
```

### 1.13.1.1.1.9 TCustomToolList.ItemRect Method

Returns rectangle occupied by the item.

```
function ItemRect(Index: integer): TRect;
```

#### 1.13.1.1.1.10 TCustomToolList.ItemsArranged Method

Called after items were rearranged by the drag&drop operations.

```
procedure ItemsArranged; virtual;
```

#### 1.13.1.1.1.11 TCustomToolList.ItemsChanged Method

Called when items were changed (any changes).

```
procedure ItemsChanged; virtual;
```

#### 1.13.1.1.1.12 TCustomToolList.ItemsHeight Method

Calculates total height of items.

```
function ItemsHeight: integer;
```

#### 1.13.1.1.1.13 TCustomToolList.MakeTopItem Method

Scrolls list to make specified item topmost item.

```
function MakeTopItem(Item: integer): Boolean;
```

#### 1.13.1.1.1.14 TCustomToolList.MakeVisible Method

Scrolls list to make specified item visible.

```
function MakeVisible(Item: integer): Boolean;
```

#### 1.13.1.1.1.15 TCustomToolList.PaintItem Method

Calls TToolListItem.Paint and allows to customize item rendering in derived classes.

```
procedure PaintItem(Item: integer; const R: TRect; State: TToolItemState); virtual;
```

#### 1.13.1.1.1.16 TCustomToolList.SelectFirstVisible Method

Selects first visible, i.e. not hidden, item.

```
function SelectFirstVisible(OnlyItem: Boolean = True): Boolean;
```

### 1.13.1.1.2 TCustomToolList Properties

#### 1.13.1.1.2.1 TCustomToolList.AllowArrange Property

Specifies whether items can be arranged by the drag&drop operations.

```
property AllowArrange: Boolean;
```

#### 1.13.1.1.2.2 TCustomToolList.AutoCollapse Property

Specifies whether all categories should be collapsed when one category is expanded or collapsed.

```
property AutoCollapse: Boolean;
```

#### 1.13.1.1.2.3 TCustomToolList.CategoryHeight Property

Specifies height of category item.

```
property CategoryHeight: integer;
```

#### 1.13.1.1.2.4 TCustomToolList.Filtered Property

Specifies whether items are filtered.

```
property Filtered: Boolean;
```

**Description**

Use Filtered property to toggle items filtration. Items (see page 121) are filtered using FilterString (see page 121) property.

**1.13.1.1.2.5 TCustomToolList.FilterString Property**

Specifies filter string which is used to test item Caption.

```
property FilterString: string;
```

**1.13.1.1.2.6 TCustomToolList.FoldingIcon Property**

Holds folding icon images.

```
property FoldingIcon: TBitmap;
```

**Description**

FoldingIcon should contain two images in a row, first - collapse icon (-), second - expand icon (+).

Color of bottom-left pixel is used as mask color.

Folding icon is initialized from resource when control is created at design time.

**1.13.1.1.2.7 TCustomToolList.HintProps Property**

Provide properties to adjust hints processing.

```
property HintProps: TechHintHelper;
```

**1.13.1.1.2.8 TCustomToolList.Images Property**

Determines which image list is associated with the tool list.

```
property Images: TCustomImageList;
```

**Description**

Use Images to provide a customized list of bitmaps that can be displayed to the left of a item's label. Individual items specify the image from this list that should appear by setting their ImageIndex property.

**1.13.1.1.2.9 TCustomToolList.InsertAtItem Property**

Specifies item index at which dragged object can be dropped.

```
property InsertAtItem: integer;
```

**1.13.1.1.2.10 TCustomToolList.ItemHeight Property**

Specifies height of the normal item.

```
property ItemHeight: integer;
```

**1.13.1.1.2.11 TCustomToolList.ItemIndex Property**

Indicates which item is selected. If no item is selected ItemIndex is equal to -1.

```
property ItemIndex: integer;
```

**1.13.1.1.2.12 TCustomToolList.Items Property**

Provides access to items displayed in tool list.

```
property Items: TToolListItems;
```

**Description**

Read Items to access the list of items that appears in the tool list. Use the methods of Items to add, insert, delete and move

items.

#### 1.13.1.1.2.13 TCustomToolList.MouseOverItem Property

Indicates item over which mouse cursor is located.

**property** MouseOverItem: integer;

#### 1.13.1.1.2.14 TCustomToolList.RightClickSelect Property

Specifies whether item can be selected by mouse right click.

**property** RightClickSelect: Boolean;

#### 1.13.1.1.2.15 TCustomToolList.RowSpace Property

Specifies space between two sequential items.

**property** RowSpace: integer;

#### 1.13.1.1.2.16 TCustomToolList.Selected Property

Currently selected item in tool list.

**property** Selected: TToolListItem;

#### 1.13.1.1.2.17 TCustomToolList.StyleCategory Property

Specifies style of category items.

**property** StyleCategory: TToolItemStyle;

#### 1.13.1.1.2.18 TCustomToolList.StyleCategoryMouseOver Property

Specifies style of category item when mouse is over it.

**property** StyleCategoryMouseOver: TToolItemStyle;

#### 1.13.1.1.2.19 TCustomToolList.StyleCategorySelected Property

Specifies style of selected category item.

**property** StyleCategorySelected: TToolItemStyle;

#### 1.13.1.1.2.20 TCustomToolList.StyleItem Property

Specifies style of tool items.

**property** StyleItem: TToolItemStyle;

#### 1.13.1.1.2.21 TCustomToolList.StyleItemMouseOver Property

Specifies style of tool item when mouse is over it.

**property** StyleItemMouseOver: TToolItemStyle;

#### 1.13.1.1.2.22 TCustomToolList.StyleItemSelected Property

Specifies style of selected tool item.

**property** StyleItemSelected: TToolItemStyle;

#### 1.13.1.1.2.23 TCustomToolList.VerticalGroups Property

Specifies whether category item should be displayed vertically along owned items.

**property** VerticalGroups: Boolean;

### 1.13.1.1.2.24 TCustomToolList.ViewOrigin Property

Specifies scrolling position of the tool list control.

**property** ViewOrigin: integer;

### 1.13.1.1.3 TCustomToolList Events

#### 1.13.1.1.3.1 TCustomToolList.OnItemArranged Event

Occurs when items order was changed.

**property** OnItemArranged: TNotifyEvent;

#### 1.13.1.1.3.2 TCustomToolList.OnItemChanged Event

Occurs when selected item is changes.

**property** OnItemChanged: TNotifyEvent;

## 1.13.1.2 TToolItemStyle Class

Holds visual properties of tool item.

### Class Hierarchy



TToolItemStyle = **class**(TPersistent);

### File

ecToolList

### Members

#### TToolItemStyle Methods

TToolItemStyle Methods	Description
Create (see page 124)	Creates and initializes a TToolItemStyle instance.
Destroy (see page 124)	Destroys an instance of TToolItemStyle.
DrawItemRect (see page 124)	Draws item frame and background.

#### TToolItemStyle Properties

TToolItemStyle Properties	Description
Alignment (see page 124)	Specifies the manner in which text is aligned within an item.
BoundPen (see page 125)	Specifies pen which is used to draw item border.
Brush (see page 125)	Specifies Brush which is used to render item background.
Font (see page 125)	Specifies font used to render item Caption.
Shape (see page 125)	Specifies shape of item frame.

#### TToolItemStyle Events

TToolItemStyle Events	Description
OnChange (see page 125)	Occurs when any properties of style are changed.

### Legend







	Constructor
	virtual
	Property
	Event













## TToolItemStyle Events

TToolItemStyle Events	Description
 OnChange (  see page 125)	Occurs when any properties of style are changed.

## TToolItemStyle Methods

TToolItemStyle Methods	Description
 Create (  see page 124)	Creates and initializes a TToolItemStyle instance.
 Destroy (  see page 124)	Destroys an instance of TToolItemStyle.
 DrawItemRect (  see page 124)	Draws item frame and background.

## TToolItemStyle Properties

TToolItemStyle Properties	Description
 Alignment (  see page 124)	Specifies the manner in which text is aligned within an item.
 BoundPen (  see page 125)	Specifies pen which is used to draw item border.
 Brush (  see page 125)	Specifies Brush which is used to render item background.
 Font (  see page 125)	Specifies font used to render item Caption.
 Shape (  see page 125)	Specifies shape of item frame.

### 1.13.1.2.1 TToolItemStyle Methods

#### 1.13.1.2.1.1 TToolItemStyle.Create Constructor

Creates and initializes a TToolItemStyle instance.

```
constructor Create;
```

##### Description

Use Create to programmatically instantiate a TToolItemStyle object.

#### 1.13.1.2.1.2 TToolItemStyle.Destroy Destructor

Destroys an instance of TToolItemStyle.

```
destructor Destroy; override;
```

##### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

#### 1.13.1.2.1.3 TToolItemStyle.DrawItemRect Method

Draws item frame and background.

```
procedure DrawItemRect(Canvas: TCanvas; const R: TRect);
```

### 1.13.1.2.2 TToolItemStyle Properties

#### 1.13.1.2.2.1 TToolItemStyle.Alignment Property

Specifies the manner in which text is aligned within an item.

```
property Alignment: TAlignment;
```

##### Description

Use the Alignment property to specify the horizontal placement of a text string within an item. Text can be aligned either to the right, left or centered.

Value	Description
taLeftJustify	Text is left justified: text line begins at the left edge of a control.

taCenter	Text is centered within a control.
taRightJustify	Text is right justified: text line ends at the right edge of a control.

1.13.1.2.2.2 TToolItemStyle.BoundPen Property

Specifies pen which is used to draw item border.

```
property BoundPen: TPen;
```

1.13.1.2.2.3 TToolItemStyle.Brush Property

Specifies Brush which is used to render item background.

```
property Brush: TBrush;
```

1.13.1.2.2.4 TToolItemStyle.Font Property

Specifies font used to render item Caption.

```
property Font: TFont;
```

1.13.1.2.2.5 TToolItemStyle.Shape Property

Specifies shape of item frame.

```
property Shape: TItemShape;
```

1.13.1.2.3 TToolItemStyle Events

1.13.1.2.3.1 TToolItemStyle.OnChange Event

Occurs when any properties of style are changed.

```
property OnChange: TNotifyEvent;
```

1.13.1.3 TToolList Class

Tool list represents item list organized in categories.

Class Hierarchy



```
TToolList = class(TCustomToolList);
```

File

ecToolList

Description

Items (see page 135) in tool list has caption, image and hint. Categories may be collapsed. Items (see page 135) may be filtered.

Members

TCustomToolList Methods

TCustomToolList Methods	Description
CollapseAll (see page 118)	Collapses all categories.
Create (see page 119)	Creates and initializes a TCustomToolList instance.
Destroy (see page 119)	Destroys an instance of TCustomToolList.
DrawItemImage (see page 119)	Draws item image.
ExpandAll (see page 119)	Expands all categories.



















































GetCategoryItem (see page 119)	Returns index of category item at the given position or above.
ItemAtPos (see page 119)	Returns items at the given position. If there are no item at the specified position function returns -1.
ItemIndexChanged (see page 119)	Called when selected item was changed.
ItemRect (see page 119)	Returns rectangle occupied by the item.
ItemsArranged (see page 120)	Called after items were rearranged by the drag&drop operations.
ItemsChanged (see page 120)	Called when items were changed (any changes).
ItemsHeight (see page 120)	Calculates total height of items.
MakeTopItem (see page 120)	Scrolls list to make specified item topmost item.
MakeVisible (see page 120)	Scrolls list to make specified item visible.
PaintItem (see page 120)	Calls TToolListItem.Paint and allows to customize item rendering in derived classes.
SelectFirstVisible (see page 120)	Selects first visible, i.e. not hidden, item.

### TCustomToolList Properties

TCustomToolList Properties	Description
AllowArrange (see page 120)	Specifies whether items can be arranged by the drag&drop operations.
AutoCollapse (see page 120)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
CategoryHeight (see page 120)	Specifies height of category item.
Filtered (see page 120)	Specifies whether items are filtered.
FilterString (see page 121)	Specifies filter string which is used to test item Caption.
FoldingIcon (see page 121)	Holds folding icon images.
HintProps (see page 121)	Provide properties to adjust hints processing.
Images (see page 121)	Determines which image list is associated with the tool list.
InsertAtItem (see page 121)	Specifies item index at which dragged object can be dropped.
ItemHeight (see page 121)	Specifies height of the normal item.
ItemIndex (see page 121)	Indicates which item is selected. If no item is selected ItemIndex is equal to -1.
Items (see page 121)	Provides access to items displayed in tool list.
MouseOverItem (see page 122)	Indicates item over which mouse cursor is located.
RightClickSelect (see page 122)	Specifies whether item can be selected by mouse right click.
RowSpace (see page 122)	Specifies space between two sequential items.
Selected (see page 122)	Currently selected item in tool list.
StyleCategory (see page 122)	Specifies style of category items.
StyleCategoryMouseOver (see page 122)	Specifies style of category item when mouse is over it.
StyleCategorySelected (see page 122)	Specifies style of selected category item.
StyleItem (see page 122)	Specifies style of tool items.
StyleItemMouseOver (see page 122)	Specifies style of tool item when mouse is over it.
StyleItemSelected (see page 122)	Specifies style of selected tool item.
VerticalGroups (see page 122)	Specifies whether category item should be displayed vertically along owned items.
ViewOrigin (see page 123)	Specifies scrolling position of the tool list control.

### TToolList Class







TToolList Class	Description
Align (see page 130)	Determines how the control aligns within its container (parent control).
AllowArrange (see page 131)	Specifies whether items can be arranged by the drag&drop operations.
Anchors (see page 131)	Specifies how the control is anchored to its parent.
AutoCollapse (see page 131)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
BevelEdges (see page 131)	Specifies which edges of the control are beveled.
BevelInner (see page 131)	Specifies the cut of the inner bevel.
BevelKind (see page 132)	Specifies the control's bevel style.
BevelOuter (see page 132)	Specifies the cut of the outer bevel.
BiDiMode (see page 132)	Specifies the bi-directional mode for the control.
CategoryHeight (see page 132)	Specifies height of category item.
Color (see page 132)	Specifies the background color of the control.
Constraints (see page 133)	Specifies the size constraints for the control.
Ctl3D (see page 133)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

 DragCursor (see page 133)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 133)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 133)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 133)	Controls whether the control responds to mouse, keyboard, and timer events.
 Filtered (see page 134)	Specifies whether items are filtered.
 FilterString (see page 134)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 134)	Holds folding icon images.
 Font (see page 134)	Controls the attributes of text written on or in the control.
 HintProps (see page 134)	Provide properties to adjust hints processing.
 Images (see page 134)	Determines which image list is associated with the tool list.
 ItemHeight (see page 134)	Specifies height of the normal item.
 ItemIndex (see page 135)	Indicates which item is selected. If no item is selected ItemIndex is equal to -1.
 Items (see page 135)	Provides access to items displayed in tool list.
 OnCanResize (see page 135)	Occurs when an attempt is made to resize the control.
 OnClick (see page 135)	Occurs when the user clicks the control.
 OnConstrainedResize (see page 135)	Adjust resize constraints.
 OnContextPopup (see page 136)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 136)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 136)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 136)	Occurs when the user drags an object over a control.
 OnEndDrag (see page 137)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 137)	Occurs when a control receives the input focus.
 OnExit (see page 137)	Occurs when the input focus shifts away from one control to another.
 OnItemArranged (see page 137)	Occurs when items order was changed.
 OnItemChanged (see page 137)	Occurs when selected item is changes.
 OnMouseDown (see page 137)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 138)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 138)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnResize (see page 138)	Occurs immediately after the control is resized.
 OnStartDrag (see page 138)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 139)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 139)	Determines where a control looks for its color information.
 ParentCtl3D (see page 139)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 139)	Determines where a control looks for its font information.
 ParentShowHint (see page 139)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 139)	Identifies the pop-up menu associated with the control.
 RightClickSelect (see page 139)	Specifies whether item can be selected by mouse right click.
 RowSpace (see page 140)	Specifies space between two sequential items.
 Selected (see page 140)	Currently selected item in tool list.
 ShowHint (see page 140)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 StyleCategory (see page 140)	Specifies style of category items.
 StyleCategoryMouseOver (see page 140)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (see page 140)	Specifies style of selected category item.
 StyleItem (see page 140)	Specifies style of tool items.
 StyleItemMouseOver (see page 140)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (see page 140)	Specifies style of selected tool item.
 TabOrder (see page 140)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 140)	Determines if the user can tab to a control.
 VerticalGroups (see page 141)	Specifies whether category item should be displayed vertically along owned items.
 Visible (see page 141)	Determines whether the component appears on screen.



**TCustomToolList Events**

TCustomToolList Events	Description
 OnItemArranged (see page 123)	Occurs when items order was changed.
 OnItemChanged (see page 123)	Occurs when selected item is changes.

















**Legend**

	Method
	virtual
	protected
	Property
	read only
	Event



















**TCustomToolList Events**

TCustomToolList Events	Description
 OnItemArranged (see page 123)	Occurs when items order was changed.
 OnItemChanged (see page 123)	Occurs when selected item is changes.

**TCustomToolList Methods**










































TCustomToolList Methods	Description
 CollapseAll (see page 118)	Collapses all categories.
 Create (see page 119)	Creates and initializes a TCustomToolList instance.
 Destroy (see page 119)	Destroys an instance of TCustomToolList.
 DrawItemImage (see page 119)	Draws item image.
 ExpandAll (see page 119)	Expands all categories.
 GetCategoryItem (see page 119)	Returns index of category item at the given position or above.
 ItemAtPos (see page 119)	Returns items at the given position. If there are no item at the specified position function returns -1.
 ItemIndexChanged (see page 119)	Called when selected item was changed.
 ItemRect (see page 119)	Returns rectangle occupied by the item.
 ItemsArranged (see page 120)	Called after items were rearranged by the drag&drop operations.
 ItemsChanged (see page 120)	Called when items were changed (any changes).
 ItemsHeight (see page 120)	Calculates total height of items.
 MakeTopItem (see page 120)	Scrolls list to make specified item topmost item.
 MakeVisible (see page 120)	Scrolls list to make specified item visible.
 PaintItem (see page 120)	Calls TToolListItem.Paint and allows to customize item rendering in derived classes.
 SelectFirstVisible (see page 120)	Selects first visible, i.e. not hidden, item.



**TCustomToolList Properties**

TCustomToolList Properties	Description
 AllowArrange (see page 120)	Specifies whether items can be arranged by the drag&drop operations.
 AutoCollapse (see page 120)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
 CategoryHeight (see page 120)	Specifies height of category item.
 Filtered (see page 120)	Specifies whether items are filtered.
 FilterString (see page 121)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 121)	Holds folding icon images.
 HintProps (see page 121)	Provide properties to adjust hints processing.
 Images (see page 121)	Determines which image list is associated with the tool list.
 InsertAtItem (see page 121)	Specifies item index at which dragged object can be dropped.
 ItemHeight (see page 121)	Specifies height of the normal item.
 ItemIndex (see page 121)	Indicates which item is selected. If no item is selected ItemIndex is equal to -1.
 Items (see page 121)	Provides access to items displayed in tool list.
 MouseOverItem (see page 122)	Indicates item over which mouse cursor is located.
 RightClickSelect (see page 122)	Specifies whether item can be selected by mouse right click.
 RowSpace (see page 122)	Specifies space between two sequential items.
  Selected (see page 122)	Currently selected item in tool list.
 StyleCategory (see page 122)	Specifies style of category items.

 StyleCategoryMouseOver (see page 122)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (see page 122)	Specifies style of selected category item.
 StyleItem (see page 122)	Specifies style of tool items.
 StyleItemMouseOver (see page 122)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (see page 122)	Specifies style of selected tool item.
 VerticalGroups (see page 122)	Specifies whether category item should be displayed vertically along owned items.
 ViewOrigin (see page 123)	Specifies scrolling position of the tool list control.

## TToolList Class

TToolList Class	Description
 Align (see page 130)	Determines how the control aligns within its container (parent control).
 AllowArrange (see page 131)	Specifies whether items can be arranged by the drag&drop operations.
 Anchors (see page 131)	Specifies how the control is anchored to its parent.
 AutoCollapse (see page 131)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
 BevelEdges (see page 131)	Specifies which edges of the control are beveled.
 BevelInner (see page 131)	Specifies the cut of the inner bevel.
 BevelKind (see page 132)	Specifies the control's bevel style.
 BevelOuter (see page 132)	Specifies the cut of the outer bevel.
 BiDiMode (see page 132)	Specifies the bi-directional mode for the control.
 CategoryHeight (see page 132)	Specifies height of category item.
 Color (see page 132)	Specifies the background color of the control.
 Constraints (see page 133)	Specifies the size constraints for the control.
 Ctl3D (see page 133)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DragCursor (see page 133)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 133)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 133)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 133)	Controls whether the control responds to mouse, keyboard, and timer events.
 Filtered (see page 134)	Specifies whether items are filtered.
 FilterString (see page 134)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 134)	Holds folding icon images.
 Font (see page 134)	Controls the attributes of text written on or in the control.
 HintProps (see page 134)	Provide properties to adjust hints processing.
 Images (see page 134)	Determines which image list is associated with the tool list.
 ItemHeight (see page 134)	Specifies height of the normal item.
 ItemIndex (see page 135)	Indicates which item is selected. If no item is selected ItemIndex is equal to -1.
 Items (see page 135)	Provides access to items displayed in tool list.
 OnCanResize (see page 135)	Occurs when an attempt is made to resize the control.
 OnClick (see page 135)	Occurs when the user clicks the control.
 OnConstrainedResize (see page 135)	Adjust resize constraints.
 OnContextPopup (see page 136)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 136)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 136)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 136)	Occurs when the user drags an object over a control.
 OnEndDrag (see page 137)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 137)	Occurs when a control receives the input focus.
 OnExit (see page 137)	Occurs when the input focus shifts away from one control to another.
 OnItemArranged (see page 137)	Occurs when items order was changed.
 OnItemChanged (see page 137)	Occurs when selected item is changes.
 OnMouseDown (see page 137)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 138)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 138)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

 OnResize (see page 138)	Occurs immediately after the control is resized.
 OnStartDrag (see page 138)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 139)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 139)	Determines where a control looks for its color information.
 ParentCtl3D (see page 139)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 139)	Determines where a control looks for its font information.
 ParentShowHint (see page 139)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 139)	Identifies the pop-up menu associated with the control.
 RightClickSelect (see page 139)	Specifies whether item can be selected by mouse right click.
 RowSpace (see page 140)	Specifies space between two sequential items.
 Selected (see page 140)	Currently selected item in tool list.
 ShowHint (see page 140)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 StyleCategory (see page 140)	Specifies style of category items.
 StyleCategoryMouseOver (see page 140)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (see page 140)	Specifies style of selected category item.
 StyleItem (see page 140)	Specifies style of tool items.
 StyleItemMouseOver (see page 140)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (see page 140)	Specifies style of selected tool item.
 TabOrder (see page 140)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 140)	Determines if the user can tab to a control.
 VerticalGroups (see page 141)	Specifies whether category item should be displayed vertically along owned items.
 Visible (see page 141)	Determines whether the component appears on screen.

### 1.13.1.3.1 TToolList Properties

#### 1.13.1.3.1.1 TToolList.Align Property

Determines how the control aligns within its container (parent control).

**property** Align;

##### Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to `allLeft`. The value of `allLeft` for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is `allNone`, which means a control remains where it is positioned on a form or panel.

**Tip:** If Align is set to `allClient`, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

**Note:** To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

### 1.13.1.3.1.2 TToolList.AllowArrange Property

Specifies whether items can be arranged by the drag&drop operations.

```
property AllowArrange: Boolean;
```

### 1.13.1.3.1.3 TToolList.Anchors Property

Specifies how the control is anchored to its parent.

```
property Anchors;
```

#### Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

**Note:** If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

### 1.13.1.3.1.4 TToolList.AutoCollapse Property

Specifies whether all categories should be collapsed when one category is expanded or collapsed.

```
property AutoCollapse: Boolean;
```

### 1.13.1.3.1.5 TToolList.BevelEdges Property

Specifies which edges of the control are beveled.

```
property BevelEdges;
```

#### Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

### 1.13.1.3.1.6 TToolList.BevelInner Property

Specifies the cut of the inner bevel.

```
property BevelInner;
```

#### Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.



### 1.13.1.3.1.7 TToolList.BevelKind Property

Specifies the control's bevel style.

```
property BevelKind;
```

#### Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

### 1.13.1.3.1.8 TToolList.BevelOuter Property

Specifies the cut of the outer bevel.

```
property BevelOuter;
```

#### Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

### 1.13.1.3.1.9 TToolList.BiDiMode Property

Specifies the bi-directional mode for the control.

```
property BiDiMode;
```

#### Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

### 1.13.1.3.1.10 TToolList.CategoryHeight Property

Specifies height of category item.

```
property CategoryHeight: integer;
```

### 1.13.1.3.1.11 TToolList.Color Property

Specifies the background color of the control.

```
property Color;
```

#### Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

### 1.13.1.3.1.12 TToolList.Constraints Property

Specifies the size constraints for the control.

```
property Constraints;
```

#### Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

**Warning:** Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

### 1.13.1.3.1.13 TToolList.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

```
property Ctl3D;
```

#### Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

### 1.13.1.3.1.14 TToolList.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

```
property DragCursor;
```

#### Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

**Note:** To make a custom cursor available for the DragCursor property, see the Cursor property.

### 1.13.1.3.1.15 TToolList.DragKind Property

Specifies whether the control is being dragged normally or for docking.

```
property DragKind;
```

#### Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

### 1.13.1.3.1.16 TToolList.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

```
property DragMode;
```

#### Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

### 1.13.1.3.1.17 TToolList.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

```
property Enabled;
```

**Description**

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

**1.13.1.3.1.18 TToolList.Filtered Property**

Specifies whether items are filtered.

```
property Filtered: Boolean;
```

**Description**

Use Filtered property to toggle items filtration. Items are filtered using FilterString property.

**1.13.1.3.1.19 TToolList.FilterString Property**

Specifies filter string which is used to test item Caption.

```
property FilterString: string;
```

**1.13.1.3.1.20 TToolList.FoldingIcon Property**

Holds folding icon images.

```
property FoldingIcon: TBitmap;
```

**Description**

FoldingIcon should contain two images in a row, first - collapse icon (-), second - expand icon (+).

Color of bottom-left pixel is used as mask color.

Folding icon is initialized from resource when control is created at design time.

**1.13.1.3.1.21 TToolList.Font Property**

Controls the attributes of text written on or in the control.

```
property Font;
```

**Description**

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

**1.13.1.3.1.22 TToolList.HintProps Property**

Provide properties to adjust hints processing.

```
property HintProps: TechHintHelper;
```

**1.13.1.3.1.23 TToolList.Images Property**

Determines which image list is associated with the tool list.

```
property Images: TCustomImageList;
```

**Description**

Use Images to provide a customized list of bitmaps that can be displayed to the left of a item's label. Individual items specify the image from this list that should appear by setting their ImageIndex property.

**1.13.1.3.1.24 TToolList.ItemHeight Property**

Specifies height of the normal item.

```
property ItemHeight: integer;
```

#### 1.13.1.3.1.25 TToolList.ItemIndex Property

Indicates which item is selected. If no item is selected ItemIndex is equal to -1.

```
property ItemIndex: integer;
```

#### 1.13.1.3.1.26 TToolList.Items Property

Provides access to items displayed in tool list.

```
property Items: TToolListItems;
```

##### Description

Read Items to access the list of items that appears in the tool list. Use the methods of Items to add, insert, delete and move items.

#### 1.13.1.3.1.27 TToolList.OnCanResize Property

Occurs when an attempt is made to resize the control.

```
property OnCanResize;
```

##### Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

#### 1.13.1.3.1.28 TToolList.OnClick Property

Occurs when the user clicks the control.

```
property OnClick;
```

#### 1.13.1.3.1.29 TToolList.OnConstrainedResize Property

Adjust resize constraints.

```
property OnConstrainedResize;
```

##### Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

##### Notes

The OnConstrainedResize handler is called immediately after the OnCanResize handler.

### 1.13.1.3.1.30 TToolList.OnContextPopup Property

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

**property** OnContextPopup;

#### Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM\_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

**Note:** Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

### 1.13.1.3.1.31 TToolList.OnDbClick Property

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

**property** OnDbClick;

#### Description

Use the OnDbClick event to respond to mouse double-clicks.

### 1.13.1.3.1.32 TToolList.OnDragDrop Property

Occurs when the user drops an object being dragged.

**property** OnDragDrop;

#### Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

### 1.13.1.3.1.33 TToolList.OnDragOver Property

Occurs when the user drags an object over a control.

**property** OnDragOver;

#### Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as

true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the `DragCursor` property for the control before the `OnDragOver` event occurs.

The `Source` is the object being dragged, the `Sender` is the potential drop or dock site, and `X` and `Y` are screen coordinates in pixels. The `State` parameter specifies how the dragged object is moving over the control.

**Note:** Within the `OnDragOver` event handler, the `Accept` parameter defaults to true. However, if an `OnDragOver` event handler is not supplied, the control rejects the dragged object, as if the `Accept` parameter were changed to false.

#### 1.13.1.3.1.34 TToolList.OnEndDrag Property

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

```
property OnEndDrag;
```

##### Description

Use the `OnEndDrag` event handler to specify any special processing that occurs when dragging stops.

#### 1.13.1.3.1.35 TToolList.OnEnter Property

Occurs when a control receives the input focus.

```
property OnEnter;
```

##### Description

Use the `OnEnter` event handler to cause any special processing to occur when a control becomes active.

The `OnEnter` event does not occur when switching between forms or between another application and the application that includes the control.

#### 1.13.1.3.1.36 TToolList.OnExit Property

Occurs when the input focus shifts away from one control to another.

```
property OnExit;
```

##### Description

Use the `OnExit` event handler to provide special processing when the control ceases to be active.

The `OnExit` event does not occur when switching between forms or between another application and your application.

#### 1.13.1.3.1.37 TToolList.OnItemArranged Property

Occurs when items order was changed.

```
property OnItemArranged: TNotifyEvent;
```

#### 1.13.1.3.1.38 TToolList.OnItemChanged Property

Occurs when selected item is changes.

```
property OnItemChanged: TNotifyEvent;
```

#### 1.13.1.3.1.39 TToolList.OnMouseDown Property

Occurs when the user presses a mouse button with the mouse pointer over a control.

```
property OnMouseDown;
```

**Description**

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

**1.13.1.3.1.40 TToolList.OnMouseMove Property**

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

**property** OnMouseMove;

**Description**

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

**1.13.1.3.1.41 TToolList.OnMouseUp Property**

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

**property** OnMouseUp;

**Description**

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

**1.13.1.3.1.42 TToolList.OnResize Property**

Occurs immediately after the control is resized.

**property** OnResize;

**Description**

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

**1.13.1.3.1.43 TToolList.OnStartDrag Property**

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

**property** OnStartDrag;

**Description**

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itse

#### 1.13.1.3.1.44 TToolList.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

```
property ParentBiDiMode;
```

#### 1.13.1.3.1.45 TToolList.ParentColor Property

Determines where a control looks for its color information.

```
property ParentColor;
```

#### 1.13.1.3.1.46 TToolList.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

```
property ParentCtl3D;
```

##### Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

#### 1.13.1.3.1.47 TToolList.ParentFont Property

Determines where a control looks for its font information.

```
property ParentFont;
```

#### 1.13.1.3.1.48 TToolList.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

```
property ParentShowHint;
```

#### 1.13.1.3.1.49 TToolList.PopupMenu Property

Identifies the pop-up menu associated with the control.

```
property PopupMenu;
```

##### Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

#### 1.13.1.3.1.50 TToolList.RightClickSelect Property

Specifies whether item can be selected by mouse right click.

```
property RightClickSelect: Boolean;
```



### 1.13.1.3.1.51 TToolList.RowSpace Property

Specifies space between two sequential items.

```
property RowSpace: integer;
```

### 1.13.1.3.1.52 TToolList.Selected Property

Currently selected item in tool list.

```
property Selected: TToolListItem;
```

### 1.13.1.3.1.53 TToolList.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

### 1.13.1.3.1.54 TToolList.StyleCategory Property

Specifies style of category items.

```
property StyleCategory: TToolItemStyle;
```

### 1.13.1.3.1.55 TToolList.StyleCategoryMouseOver Property

Specifies style of category item when mouse is over it.

```
property StyleCategoryMouseOver: TToolItemStyle;
```

### 1.13.1.3.1.56 TToolList.StyleCategorySelected Property

Specifies style of selected category item.

```
property StyleCategorySelected: TToolItemStyle;
```

### 1.13.1.3.1.57 TToolList.StyleItem Property

Specifies style of tool items.

```
property StyleItem: TToolItemStyle;
```

### 1.13.1.3.1.58 TToolList.StyleItemMouseOver Property

Specifies style of tool item when mouse is over it.

```
property StyleItemMouseOver: TToolItemStyle;
```

### 1.13.1.3.1.59 TToolList.StyleItemSelected Property

Specifies style of selected tool item.

```
property StyleItemSelected: TToolItemStyle;
```

### 1.13.1.3.1.60 TToolList.TabOrder Property

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

### 1.13.1.3.1.61 TToolList.TabStop Property

Determines if the user can tab to a control.

```
property TabStop;
```

#### Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

1.13.1.3.1.62 TToolList.VerticalGroups Property

Specifies whether category item should be displayed vertically along owned items.

```
property VerticalGroups: Boolean;
```

1.13.1.3.1.63 TToolList.Visible Property

Determines whether the component appears on screen.

```
property Visible;
```

Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

1.13.1.4 TToolListItem Class

Represents tool item used in tool lists ( see page 116).

Class Hierarchy



```
TToolListItem = class(TCollectionItem);
```

File

ecToolList

Description

Item class is common for normal and category items. They are differ from each other by the IsCategory ( see page 142) property.

Members



















TToolListItem Properties

TToolListItem Properties	Description
Caption ( see page 142)	Specifies Caption of the tool item.
Expanded ( see page 142)	Specifies whether category item is expanded.
Hint ( see page 142)	Contains the text string that can appear when the user moves the mouse over the item.
ImageIndex ( see page 142)	Determines index in an image list.
IsCategory ( see page 142)	Specifies whether tool item is category item. If this property is False - item is normal.
Tag ( see page 142)	Stores an integer value as part of an item.
ToolList ( see page 143)	Provides reference to owner tool list.
Visible ( see page 143)	Indicates whether item is visible, i.e. not hidden by the filter.

Legend

	Property
	protected
	read only

**TToolListItem Properties**

<b>TToolListItem Properties</b>	<b>Description</b>
 Caption (  see page 142)	Specifies Caption of the tool item.
 Expanded (  see page 142)	Specifies whether category item is expanded.
 Hint (  see page 142)	Contains the text string that can appear when the user moves the mouse over the item.
 ImageIndex (  see page 142)	Determines index in an image list.
 IsCategory (  see page 142)	Specifies whether tool item is category item. If this property is False - item is normal.
 Tag (  see page 142)	Stores an integer value as part of an item.
  ToolList (  see page 143)	Provides reference to owner tool list.
  Visible (  see page 143)	Indicates whether item is visible, i.e. not hidden by the filter.

**1.13.1.4.1 TToolListItem Properties****1.13.1.4.1.1 TToolListItem.Caption Property**

Specifies Caption of the tool item.

```
property Caption: WideString;
```

**1.13.1.4.1.2 TToolListItem.Expanded Property**

Specifies whether category item is expanded.

```
property Expanded: Boolean;
```

**Description**

Expanded property only takes effect for category items, i.e. when IsCategory ( see page 142) is True.

**1.13.1.4.1.3 TToolListItem.Hint Property**

Contains the text string that can appear when the user moves the mouse over the item.

```
property Hint: WideString;
```

**1.13.1.4.1.4 TToolListItem.ImageIndex Property**

Determines index in an image list.

```
property ImageIndex: TImageIndex;
```

**Description**

Set ImageIndex to associate an item with an image in the Images property of TCustomToolList ( see page 116).

**1.13.1.4.1.5 TToolListItem.IsCategory Property**

Specifies whether tool item is category item. If this property is False - item is normal.

```
property IsCategory: Boolean;
```

**1.13.1.4.1.6 TToolListItem.Tag Property**

Stores an integer value as part of an item.

```
property Tag: integer;
```

**Description**

Tag has no predefined meaning. The Tag property is provided for the convenience of developers. It can be used for storing an additional integer value or it can be typecast to any 32-bit value.

### 1.13.1.4.1.7 TToolListItem.ToolList Property

Provides reference to owner tool list.

```
property ToolList: TCustomToolList;
```

### 1.13.1.4.1.8 TToolListItem.Visible Property

Indicates whether item is visible, i.e. not hidden by the filter.

```
property Visible: Boolean;
```

## 1.13.1.5 TToolListItems Class

Collection of tool list items.

### Class Hierarchy





```
TToolListItems = class(TOwnedCollection);
```

### File



ecToolList

### Members


#### TToolListItems Properties

TToolListItems Properties	Description
  Items  see page 143	Lists the items in the collection.

### Legend

	Property
	read only

#### TToolListItems Properties

TToolListItems Properties	Description
  Items  see page 143	Lists the items in the collection.

### 1.13.1.5.1 TToolListItems Properties

#### 1.13.1.5.1.1 TToolListItems.Items Property

Lists the items in the collection.

```
property Items [Index: integer]: TToolListItem;
```

## 1.13.2 Structs, Records, Enums

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

### Enumerations

Enumeration	Description
 TItemShape  see page 144	Specifies items shape.

### Legend

	Enumeration
-------------------------------------------------------------------------------------	-------------

### 1.13.2.1 ecToolList.TItemShape Enumeration

Specifies items shape.

```
TItemShape = (  
    isRectangle,  
    isRoundRect  
);
```

**File**

ecToolList

**Members**

Members	Description
isRectangle	Rectangle item shape.
isRoundRect	Rectangle shape with rounded corners.

### 1.13.3 Types

The following table lists types in this documentation.

**Types**

Type	Description
TToolItemState (see page 144)	Specifies tool item state which is used for item rendering.

#### 1.13.3.1 ecToolList.TToolItemState Type

Specifies tool item state which is used for item rendering.

```
TToolItemState = set of (tisSelected, tisPressed, tisMouseOver);
```

**File**

ecToolList

## 1.14 ed\_DsnBase Namespace

### 1.14.1 Classes

The following table lists classes in this documentation.

**Classes**

Class	Description
TBaseDesigner (see page 144)	Base designer component.

#### 1.14.1.1 TBaseDesigner Class

Base designer component.

## Class Hierarchy



```
TBaseDesigner = class(TComponent);
```

## File

ed\_DsnBase

## Description

TBaseDesigner dispatches messages and controls designer hints.

## Members

### TBaseDesigner Methods

TBaseDesigner Methods	Description
CanProcessNCMessages (see page 146)	Called to check processing of non client mouse messages.
Client2Screen (see page 147)	Corrected version of ClientToScreen function to support RTL.
ClientOrg (see page 147)	Corrected version of ClientOrigin function to support RTL.
Create (see page 147)	Creates and initializes a TBaseDesigner instance.
DesignState (see page 147)	Indicates whether component in design or loading state. At design time (in Delphi IDE) and during loading no activation occurs.
Destroy (see page 147)	Destroys an instance of TBaseDesigner.
DoObjectHint (see page 147)	Called to show hint for current design state.
DragDrop (see page 147)	OnDragDrop (see page 152) event dispatcher.
DragOver (see page 148)	OnDragOver (see page 152) event dispatcher.
IsRTL (see page 148)	Returns true when control is right-to-left.
KeyDown (see page 148)	Respond to key press events.
KeyPress (see page 149)	Respond to keyboard input.
KeyUp (see page 149)	Respond to released key.
Loaded (see page 149)	Initializes the component after the form file has been read into memory.
MouseDown (see page 150)	OnMouseDown (see page 154) event dispatcher.
MouseMove (see page 150)	OnMouseMove (see page 154) event dispatcher.
MouseUp (see page 150)	OnMouseUp (see page 154) event dispatcher.
ProcessMessage (see page 151)	Translates messages of all managed by designer controls.
ResetHint (see page 151)	Resets hint for another component
Screen2Client (see page 151)	Corrected version of ScreenToClient function to support RTL.
SetActive (see page 151)	Set method for Active (see page 151) property.
ShowHint (see page 151)	Indicates whether hints should be shown for controls in the Designer.






### TBaseDesigner Properties

TBaseDesigner Properties	Description
Active (see page 151)	Switches target component between design and run-time modes
HintObject (see page 152)	Specifies object for which hint was activated.
ShowHints (see page 152)	Specifies showing of design hints.











### TBaseDesigner Events

TBaseDesigner Events	Description
OnActiveChanged (see page 152)	Occurs when the Active (see page 151) property of the TzCustomFormDesigner changes
OnDragDrop (see page 152)	Occurs when the user drops an object being dragged.
OnDragOver (see page 152)	Occurs when the user drags an object over a control.
OnHandleControlMessage (see page 153)	Occurs on any message sent to managed controls.
OnKeyDown (see page 153)	Occurs only at design mode when user presses down any key.
OnKeyPress (see page 153)	Occurs only at design mode when key pressed.
OnKeyUp (see page 154)	Occurs only at design mode when user releases key that has been pressed.
OnMouseDown (see page 154)	Occurs only at design mode when user presses mouse button.
OnMouseMove (see page 154)	Occurs only at design mode when user moves mouse.
OnMouseUp (see page 154)	Occurs only at design mode when user releases mouse button.
























**Legend**

	Method
	protected
	virtual
	Property
	Event




**TBaseDesigner Events**

TBaseDesigner Events	Description
 OnActiveChanged ( <a href="#">see page 152</a> )	Occurs when the Active ( <a href="#">see page 151</a> ) property of the TzCustomFormDesigner changes
 OnDragDrop ( <a href="#">see page 152</a> )	Occurs when the user drops an object being dragged.
 OnDragOver ( <a href="#">see page 152</a> )	Occurs when the user drags an object over a control.
 OnHandleControlMessage ( <a href="#">see page 153</a> )	Occurs on any message sent to managed controls.
 OnKeyDown ( <a href="#">see page 153</a> )	Occurs only at design mode when user presses down any key.
 OnKeyPress ( <a href="#">see page 153</a> )	Occurs only at design mode when key pressed.
 OnKeyUp ( <a href="#">see page 154</a> )	Occurs only at design mode when user releases key that has been pressed.
 OnMouseDown ( <a href="#">see page 154</a> )	Occurs only at design mode when user presses mouse button.
 OnMouseMove ( <a href="#">see page 154</a> )	Occurs only at design mode when user moves mouse.
 OnMouseUp ( <a href="#">see page 154</a> )	Occurs only at design mode when user releases mouse button.

**TBaseDesigner Methods**

TBaseDesigner Methods	Description
  CanProcessNCMessages ( <a href="#">see page 146</a> )	Called to check processing of non client mouse messages.
 Client2Screen ( <a href="#">see page 147</a> )	Corrected version of ClientToScreen function to support RTL.
 ClientOrg ( <a href="#">see page 147</a> )	Corrected version of ClientOrigin function to support RTL.
 Create ( <a href="#">see page 147</a> )	Creates and initializes a TBaseDesigner instance.
 DesignState ( <a href="#">see page 147</a> )	Indicates whether component in design or loading state. At design time (in Delphi IDE) and during loading no activation occurs.
 Destroy ( <a href="#">see page 147</a> )	Destroys an instance of TBaseDesigner.
 DoObjectHint ( <a href="#">see page 147</a> )	Called to show hint for current design state.
 DragDrop ( <a href="#">see page 147</a> )	OnDragDrop ( <a href="#">see page 152</a> ) event dispatcher.
 DragOver ( <a href="#">see page 148</a> )	OnDragOver ( <a href="#">see page 152</a> ) event dispatcher.
 IsRTL ( <a href="#">see page 148</a> )	Returns true when control is right-to-left.
 KeyDown ( <a href="#">see page 148</a> )	Respond to key press events.
 KeyPress ( <a href="#">see page 149</a> )	Respond to keyboard input.
 KeyUp ( <a href="#">see page 149</a> )	Respond to released key.
 Loaded ( <a href="#">see page 149</a> )	Initializes the component after the form file has been read into memory.
 MouseDown ( <a href="#">see page 150</a> )	OnMouseDown ( <a href="#">see page 154</a> ) event dispatcher.
 MouseMove ( <a href="#">see page 150</a> )	OnMouseMove ( <a href="#">see page 154</a> ) event dispatcher.
 MouseUp ( <a href="#">see page 150</a> )	OnMouseUp ( <a href="#">see page 154</a> ) event dispatcher.
 ProcessMessage ( <a href="#">see page 151</a> )	Translates messages of all managed by designer controls.
 ResetHint ( <a href="#">see page 151</a> )	Resets hint for another component
 Screen2Client ( <a href="#">see page 151</a> )	Corrected version of ScreenToClient function to support RTL.
 SetActive ( <a href="#">see page 151</a> )	Set method for Active ( <a href="#">see page 151</a> ) property.
 ShowHint ( <a href="#">see page 151</a> )	Indicates whether hints should be shown for controls in the Designer.

**TBaseDesigner Properties**

TBaseDesigner Properties	Description
 Active ( <a href="#">see page 151</a> )	Switches target component between design and run-time modes
 HintObject ( <a href="#">see page 152</a> )	Specifies object for which hint was activated.
 ShowHints ( <a href="#">see page 152</a> )	Specifies showing of design hints.

**1.14.1.1.1 TBaseDesigner Methods****1.14.1.1.1.1 TBaseDesigner.CanProcessNCMessages Method**

Called to check processing of non client mouse messages.

```
function CanProcessNCMessages(Sender: TControl): Boolean; virtual;
```

**Description**

Called to check whether mouse messages to not client area should be processed as normal mouse messages. By default, WM\_NCXXX messages processed for all controls except forms.

**1.14.1.1.1.2 TBaseDesigner.Client2Screen Method**

Corrected version of ClientToScreen function to support RTL.

```
function Client2Screen(Ctl: TControl; Point: TPoint): TPoint;
```

**1.14.1.1.1.3 TBaseDesigner.ClientOrg Method**

Corrected version of ClientOrigin function to support RTL.

```
function ClientOrg(Ctl: TControl): TPoint;
```

**1.14.1.1.1.4 TBaseDesigner.Create Constructor**

Creates and initializes a TBaseDesigner instance.

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

**Description**

Use Create to programmatically instantiate a TBaseDesigner component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

**1.14.1.1.1.5 TBaseDesigner.DesignState Method**

```
function DesignState: Boolean;
```

**Description**

Indicates whether component in design or loading state. At design time (in Delphi IDE) and during loading no activation occurs.

**1.14.1.1.1.6 TBaseDesigner.Destroy Destructor**

Destroys an instance of TBaseDesigner.

```
destructor Destroy; override;
```

**Description**

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

**1.14.1.1.1.7 TBaseDesigner.DoObjectHint Method**

Called to show hint for current design state.

```
procedure DoObjectHint; virtual;
```

**Description**

DoObjectHint method updates hint window content and position.

**1.14.1.1.1.8 TBaseDesigner.DragDrop Method**

OnDragDrop (see page 152) event dispatcher.

```
procedure DragDrop(Sender: TObject; Source: TObject; X: Integer; Y: Integer); virtual;
```



**Description**

Override DragDrop to add additional code that executes before the OnDragDrop (see page 152) event handler is called.

The Source parameter is the object that was dropped onto the control Sender. The X and Y parameters are the mouse coordinates where the object was dropped.

**1.14.1.1.1.9 TBaseDesigner.DragOver Method**

OnDragOver (see page 152) event dispatcher.

```
function DragOver(Sender: TObject; Source: TObject; X: Integer; Y: Integer; State: TDragState): Boolean; virtual;
```

**Description**

Override DragOver to add additional code that executes before the OnDragOver (see page 152) event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

**1.14.1.1.1.10 TBaseDesigner.IsRTL Method**

Returns true when control is right-to-left.

```
function IsRTL(Ctl: TControl): Boolean;
```

**1.14.1.1.1.11 TBaseDesigner.KeyDown Method**

Respond to key press events.

```
procedure KeyDown(var Key: Word; Shift: TShiftState); virtual;
```

**Description**

When a windowed control or design surface receives a key-down message (WM\_KEYDOWN) from Windows, its message handler calls KeyDown, passing the key code and shift-key state in the Key and Shift parameters, respectively.

KeyDown calls any event handler attached to the OnKeyDown (see page 153) event. Override KeyDown to provide other responses in addition to the event-handler call.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use WinAPI virtual key codes to determine the key pressed. For more information, search for virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

Either KeyDown or the OnKeyDown (see page 153) event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

### 1.14.1.1.1.12 TBaseDesigner.KeyPress Method

Respond to keyboard input.

```
procedure KeyPress(var Key: Char); virtual;
```

#### Description

When a windowed control or design surface receives a key-press message (WM\_CHAR) from Windows, its message handler calls KeyPress, passing the key code in the Key parameter.

KeyPress calls any event handler attached to the OnKeyPress (see page 153) event. Override KeyPress to provide other responses in addition to the event-handler call.

Either KeyPress or the OnKeyPress (see page 153) event handler it calls can suppress further processing of a character by setting the Key parameter to zero.

Note: The Key parameter is the character represented by the key that is pressed, not a Windows virtual key code.

### 1.14.1.1.1.13 TBaseDesigner.KeyUp Method

Respond to released key.

```
procedure KeyUp(var Key: Word; Shift: TShiftState); virtual;
```

#### Description

When a windowed control receives a key-up message (WM\_KEYUP) from Windows, its message handler calls KeyUp, passing the key code and shift-key state to KeyUp in the Key and Shift parameters, respectively.

KeyUp calls any event handler attached to the OnKeyUp (see page 154) event. Override KeyUp to provide other responses in addition to the event-handler call.

Either KeyUp or the OnKeyUp (see page 154) event handler it calls can suppress further processing of a key by setting the Key parameter to zero.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use WinAPI virtual key codes to determine the key pressed. For more information, search virtual key codes in the Win32 Developer's Reference (WIN32.HLP).

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

### 1.14.1.1.1.14 TBaseDesigner.Loaded Method

Initializes the component after the form file has been read into memory.

```
procedure Loaded; override;
```

#### Description

It is overridden procedure from TComponent.Loaded

First it calls inherited then set Active (see page 151) to reading state.

#### 1.14.1.1.1.15 TBaseDesigner.MouseDown Method

OnMouseDown (see page 154) event dispatcher.

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
virtual;
```

##### Description

Override the protected MouseDown method to provide other responses in addition to calling the OnMouseDown (see page 154) event handler when the user presses mouse button while the cursor's hotspot is over the managed control.

A designer calls MouseDown in response to any of the Windows mouse-down messages (WM\_LBUTTONDOWN, WM\_MBUTTONDOWN, WM\_RBUTTONDOWN) sent to managed controls, decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

X,Y coordinates are relative to client origin of the root pane FRootPane.

#### 1.14.1.1.1.16 TBaseDesigner.MouseMove Method

OnMouseMove (see page 154) event dispatcher.

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); virtual;
```

##### Description

Override the protected MouseMove method to provide other responses in addition to calling the OnMouseMove (see page 154) event handler when the user moves the mouse.

A control calls MouseMove in response to any of the Windows mouse-move messages (WM\_MOUSEMOVE), decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively.

As the mouse cursor moves across a control, this method is called repeatedly. Each time it is called, it is with the new coordinates that reflect the continuous path of the mouse cursor across the screen real estate covered by the control's visual representation

X,Y coordinates are relative to client origin of the root pane FRootPane.

#### 1.14.1.1.1.17 TBaseDesigner.MouseUp Method

OnMouseUp (see page 154) event dispatcher.

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
virtual;
```

##### Description

Override the protected MouseUp method to provide other responses in addition to calling the OnMouseUp (see page 154) event handler when the user releases a previously pressed mouse button while the cursor's hotspot is over the control.

A designer calls MouseUp in response to any of the Windows mouse-up messages (WM\_LBUTTONUP, WM\_MBUTTONUP, WM\_RBUTTONUP) sent to managed controls, decoding the message parameters into the shift-key state and position, which it passes in the Shift, X, and Y parameters, respectively. The value of the Button parameter indicates which mouse button was released: left, right, or middle

X,Y coordinates are relative to client origin of the root pane FRootPane.

#### 1.14.1.1.1.18 TBaseDesigner.ProcessMessage Method

Translates messages of all managed by designer controls.

```
function ProcessMessage(Sender: TControl; var Message: TMessage): Boolean;
```

##### Description

#### 1.14.1.1.1.19 TBaseDesigner.ResetHint Method

Resets hint for another component

```
procedure ResetHint;
```

##### Description

#### 1.14.1.1.1.20 TBaseDesigner.Screen2Client Method

Corrected version of ScreenToClient function to support RTL.

```
function Screen2Client(Ctl: TControl; Point: TPoint): TPoint;
```

#### 1.14.1.1.1.21 TBaseDesigner.SetActive Method

Set method for Active (☑ see page 151) property.

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

##### Description

Write overridden method to process actions for designer activation/deactivation.

#### 1.14.1.1.1.22 TBaseDesigner.ShowHint Method

Indicates whether hints should be shown for controls in the Designer.

```
procedure ShowHint(const AHint: string);
```

##### Description

Set ShowHints (☑ see page 152) to False if you don't want hints shown for controls on the designed form.

### 1.14.1.1.2 TBaseDesigner Properties

#### 1.14.1.1.2.1 TBaseDesigner.Active Property

Switches target component between design and run-time modes

```
property Active: Boolean;
```

##### Description

Active is one of main properties of TzCustomFormDesigner.

It switches target component (it must be TWinControl descendant) between design and run-time mode.

When in design mode all of parents controls are in design mode too so user can manipulate all of this properties.

Manipulating with nonvial components depends on AllowComponents property.

### 1.14.1.1.2.2 TBaseDesigner.HintObject Property

Specifies object for which hint was activated.

```
property HintObject: TObject;
```

#### Description

### 1.14.1.1.2.3 TBaseDesigner.ShowHints Property

Specifies showing of design hints.

```
property ShowHints: Boolean;
```

#### Description

Use this property to enable/disable designer's hints.

## 1.14.1.1.3 TBaseDesigner Events

### 1.14.1.1.3.1 TBaseDesigner.OnActiveChanged Event

Occurs when the Active (see page 151) property of the TzCustomFormDesigner changes

```
property OnActiveChanged: TNotifyEvent;
```

#### Description

Write an OnActiveChange event handler to take specific action immediately after the TzCustomFormDesigner changes its Active (see page 151) property. For example user can prohibit saving or loading form being designed while Active = True.

The Sender parameter is the object whose event handler is called.

### 1.14.1.1.3.2 TBaseDesigner.OnDragDrop Event

Occurs when the user drops an object being dragged.

```
property OnDragDrop: TDragDropEvent;
```

#### Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control

### 1.14.1.1.3.3 TBaseDesigner.OnDragOver Event

Occurs when the user drags an object over a control.

```
property OnDragOver: TDragOverEvent;
```

#### Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop target, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

#### 1.14.1.1.3.4 TBaseDesigner.OnHandleControlMessage Event

Occurs on any message sent to managed controls.

**property** OnHandleControlMessage: THandleControlMessage;

##### Description

Write this event handler to process messages.

Sender - designer component;

Control - control to which message was sent;

Message - message;

Handled - handling flag. Set this flag to True to abort following message processing.

#### 1.14.1.1.3.5 TBaseDesigner.OnKeyDown Event

Occurs only at design mode when user presses down any key.

**property** OnKeyDown: TKeyEvent;

##### Description

It is the same as standard TWinControl.OnKeyDown event.

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys, and pressed mouse buttons.

See TWinControl.OnKeyDown for details

#### 1.14.1.1.3.6 TBaseDesigner.OnKeyPress Event

Occurs only at design mode when key pressed.

**property** OnKeyPress: TKeyPressEvent;

##### Description

It is the same as standard TWinControl.OnKeyPress event.

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

See TWinControl.OnKeyPress for details

#### 1.14.1.1.3.7 TBaseDesigner.OnKeyUp Event

Occurs only at design mode when user releases key that has been pressed.

**property** OnKeyUp: TKeyEvent;

##### Description

It is the same as standard TWinControl.OnKeyUp event.

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

See TWinControl.OnKeyUp for details

#### 1.14.1.1.3.8 TBaseDesigner.OnMouseDown Event

Occurs only at design mode when user presses mouse button.

**property** OnMouseDown: TMouseEvent;

##### Description

It is the same as standard TControl.OnMouseDown event.

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

See TControl.OnMouseDown for details.

#### 1.14.1.1.3.9 TBaseDesigner.OnMouseMove Event

Occurs only at design mode when user moves mouse.

**property** OnMouseMove: TMouseMoveEvent;

##### Description

It is the same as standard TControl.OnMouseMove event.

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

See TControl.OnMouseMove for details.

#### 1.14.1.1.3.10 TBaseDesigner.OnMouseUp Event

Occurs only at design mode when user releases mouse button.

**property** OnMouseUp: TMouseEvent;

##### Description

It is the same as standard TControl.OnMouseUp event.

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

See TControl.OnMouseUp for details.

## 1.14.2 Structs, Records, Enums

### 1.14.2.1 ed\_DsnBase.TDesignOperation Enumeration

```
TDesignOperation = (  
    doSize,  
    doMove,  
    doSelect,  
    doDelete,  
    doInsert  
);
```

**File**

ed\_DsnBase

**Members**

Members	Description
doSize	Resize of selected control
doMove	Move selection
doSelect	Select controls within rectangle
doDelete	Delete selected components.
doInsert	Insert new component

**Description**

Design operations.

### 1.14.2.2 ed\_DsnBase.TDsnDragState Enumeration

```
TDsnDragState = (  
    dsNone,  
    dsRect,  
    dsSelMove,  
    dsResize,  
    dsInsert  
);
```

**File**

ed\_DsnBase

**Members**

Members	Description
dsNone	No drag operation
dsRect	Dragging selection rectangle
dsSelMove	Selecting and moving controls
dsResize	Resizing selected control
dsInsert	Inserting new component.

**Description**

State of drag operation in Designer



### 1.14.3 Types

The following table lists types in this documentation.

Types

Type	Description
TDesignOperations (🔗 see page 156)	A set of design operations.
THandleControlMessage (🔗 see page 156)	See TBaseDesigner.OnHandleControlMessage (🔗 see page 153).

#### 1.14.3.1 ed\_DsnBase.TDesignOperations Type

```
TDesignOperations = set of TDesignOperation;
```

File

ed\_DsnBase

Description

A set of design operations.

#### 1.14.3.2 ed\_DsnBase.THandleControlMessage Type

```
THandleControlMessage = procedure (Sender: TObject; Control: TControl; var Message: TMessage; var Handled: Boolean) of object;
```

File

ed\_DsnBase

Description

See TBaseDesigner.OnHandleControlMessage (🔗 see page 153).

## 1.15 ed\_Designer Namespace

### 1.15.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TControlGroups (🔗 see page 156)	Manages designer's groups of controls.
TPasteInfo (🔗 see page 159)	TPasteInfo is used in paste operation from the buffer
TzCustomFormDesigner (🔗 see page 161)	TzCustomFormDesigner is main library component.
TzFormDesigner (🔗 see page 201)	TzFormDesigner is main library component.

#### 1.15.1.1 TControlGroups Class

Manages designer's groups of controls.

## Class Hierarchy



```
TControlGroups = class(TPersistent);
```

## File

ed\_Designer

## Description

Grouping of controls effects on control selection. When one control of group is selected - all other control in this group are selected too.

## Members

### TControlGroups Methods

TControlGroups Methods	Description
Clear (see page 158)	Removes all group information.
Create (see page 158)	Creates and initializes a TControlGroups instance.
Destroy (see page 158)	Destroys an instance of TControlGroups.
GroupControls (see page 158)	Moves controls from list in single group. List object is deleted or saved in this method and must not be deleted outside.
GroupForControl (see page 158)	Returns list of objects to which control Ctl belongs.
GroupSelected (see page 158)	Groups (see page 158) selected controls.
UnGroup (see page 158)	Removes group with index GroupIndex.
UnGroupSelected (see page 158)	Removes all groups which contain selected controls.

### TControlGroups Properties

TControlGroups Properties	Description
Count (see page 158)	Specifies number of groups.
Groups (see page 158)	Provides index access to groups. Each group is list of TControl references. <b>Note:</b> do not destroy (see page 158) these lists. They are managed by the TControlGroups object.

## Legend

	Method
	virtual
	Property
	read only

### TControlGroups Methods

TControlGroups Methods	Description
Clear (see page 158)	Removes all group information.
Create (see page 158)	Creates and initializes a TControlGroups instance.
Destroy (see page 158)	Destroys an instance of TControlGroups.
GroupControls (see page 158)	Moves controls from list in single group. List object is deleted or saved in this method and must not be deleted outside.
GroupForControl (see page 158)	Returns list of objects to which control Ctl belongs.
GroupSelected (see page 158)	Groups (see page 158) selected controls.
UnGroup (see page 158)	Removes group with index GroupIndex.
UnGroupSelected (see page 158)	Removes all groups which contain selected controls.

### TControlGroups Properties

TControlGroups Properties	Description
Count (see page 158)	Specifies number of groups.
Groups (see page 158)	Provides index access to groups. Each group is list of TControl references. <b>Note:</b> do not destroy (see page 158) these lists. They are managed by the TControlGroups object.

## 1.15.1.1.1 TControlGroups Methods

#### 1.15.1.1.1.1 TControlGroups.Clear Method

Removes all group information.

```
procedure Clear;
```

#### 1.15.1.1.1.2 TControlGroups.Create Constructor

Creates and initializes a TControlGroups instance.

```
constructor Create(AOwner: TzCustomFormDesigner);
```

##### Description

Use Create to programmatically instantiate a TControlGroups object.

#### 1.15.1.1.1.3 TControlGroups.Destroy Destructor

Destroys an instance of TControlGroups.

```
destructor Destroy; override;
```

##### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

#### 1.15.1.1.1.4 TControlGroups.GroupControls Method

Moves controls from list in single group. List object is deleted or saved in this method and must not be deleted outside.

```
procedure GroupControls(List: TList);
```

#### 1.15.1.1.1.5 TControlGroups.GroupForControl Method

Returns list of objects to which control Ctl belongs.

```
function GroupForControl(Ctl: TObject): TList;
```

#### 1.15.1.1.1.6 TControlGroups.GroupSelected Method

Groups (see page 158) selected controls.

```
procedure GroupSelected;
```

#### 1.15.1.1.1.7 TControlGroups.UnGroup Method

Removes group with index GroupIndex.

```
procedure UnGroup(GroupIndex: integer);
```

#### 1.15.1.1.1.8 TControlGroups.UnGroupSelected Method

Removes all groups which contain selected controls.

```
procedure UnGroupSelected;
```

### 1.15.1.1.2 TControlGroups Properties

#### 1.15.1.1.2.1 TControlGroups.Count Property

Specifies number of groups.

```
property Count: integer;
```

#### 1.15.1.1.2.2 TControlGroups.Groups Property

Provides index access to groups. Each group is list of TControl references.

**Note:** do not destroy (see page 158) these lists. They are managed by the TControlGroups (see page 156) object.

```
property Groups [Index: integer]: TList;
```

## 1.15.1.2 TPasteInfo Class

TPasteInfo is used in paste operation from the buffer

### Class Hierarchy

ed\_Designer.TPasteInfo

```
TPasteInfo = class;
```

### File

ed\_Designer





### Description

TPasteInfo store information about array of controls placed in the buffer so that this controls are pasted onto parent with correct offset.

With object of this class pasting procedure are very similar to standard Borland IDE pasting one.

### Members





#### TPasteInfo Methods

TPasteInfo Methods	Description
 Create (see page 159)	Creates and initializes a TPasteInfo instance.
 Destroy (see page 160)	Destroys object and releases all the resources
 IncForParent (see page 160)	Calculates and returns actual offset for this Parent
 Init (see page 160)	Initialization of TPasteInfo object.





#### TPasteInfo Properties

TPasteInfo Properties	Description
 CurrOffset (see page 160)	CurrOffset keeps offset, in pixels, for next pasting procedure.


### Legend

	Constructor
	virtual
	Property
	read only

#### TPasteInfo Methods

TPasteInfo Methods	Description
 Create (see page 159)	Creates and initializes a TPasteInfo instance.
 Destroy (see page 160)	Destroys object and releases all the resources
 IncForParent (see page 160)	Calculates and returns actual offset for this Parent
 Init (see page 160)	Initialization of TPasteInfo object.

#### TPasteInfo Properties

TPasteInfo Properties	Description
 CurrOffset (see page 160)	CurrOffset keeps offset, in pixels, for next pasting procedure.

## 1.15.1.2.1 TPasteInfo Methods

### 1.15.1.2.1.1 TPasteInfo.Create Constructor

Creates and initializes a TPasteInfo (see page 159) instance.

```
constructor Create;
```

**Description**

Create initializes inner fields for the TPasteInfo (see page 159) object.

**1.15.1.2.1.2 TPasteInfo.Destroy Destructor**

Destroys object and releases all the resources

```
destructor Destroy; override;
```

**Description****1.15.1.2.1.3 TPasteInfo.IncForParent Method**

Calculates and returns actual offset for this Parent

```
function IncForParent(Parent: TWinControl; Offset: Integer): Integer;
```

**Description**

IncForParent calculates and returns offset for current Parent.

It stores offset for each parent it has applied so that offset increment is smart.

**Parent** is pointer to wincontrol components are placed onto.

**Offset** is number of pixels, for what user intends to place component off as it would be single one.

This method calculates offset depending on count of buffer's components and parent so that pasted components do not overlap existing ones.

**1.15.1.2.1.4 TPasteInfo.Init Method**

Initialization of TPasteInfo (see page 159) object.

```
procedure Init(SelCount: Integer; InitParent: TWinControl; BufType: TBufferizedType;  
Offset: Integer);
```

**Description**

User must initialize object TPasteInfo (see page 159) at point of copying/cutting components in buffer with correct parameters.

**SelCount** is number of selected (and correspondingly copied or cut into the buffer) components.

**InitParent** is a pointer to WinControl those components are taken from.

**BufType** determines whether components are cut or copied.

**Offset** is a number of pixels from initial offset (it may be grid step).

**1.15.1.2.2 TPasteInfo Properties****1.15.1.2.2.1 TPasteInfo.CurrOffset Property**

CurrOffset keeps offset, in pixels, for next pasting procedure.

```
property CurrOffset: Integer;
```

**Description**

This property stores current offset, in pixels, for next pasting procedure.

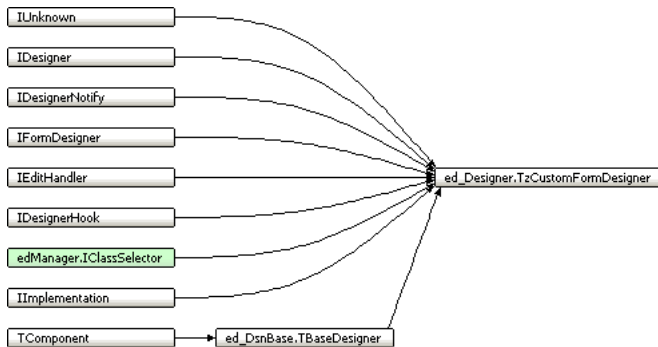
Read-only property.

It depends on count of components in buffer and parent control used for pasting.

### 1.15.1.3 TzCustomFormDesigner Class

TzCustomFormDesigner is main library component.

#### Class Hierarchy



```
TzCustomFormDesigner = class(TBaseDesigner, IUnknown, IDesigner, IDesignerNotify,
IFormDesigner, IEditHandler, IDesignerHook, IClassSelector, IImplementation);
```

#### File

ed\_Designer

#### Description

TzCustomFormDesigner provides full-sized functionality to manipulate with controls at run-time as if it was to be in design-time IDE.

TzCustomFormDesigner implements all the required design-time interfaces for this purpose and propose a set of properties, methods and events to create (see page 174) design-like environment at runtime.

With TzCustomFormDesigner user may design forms, data-modules, report sheets and so on.

It uses all the standard Borland property and component editors.

#### Members

##### TBaseDesigner Methods

TBaseDesigner Methods	Description
CanProcessNCMessages (see page 146)	Called to check processing of non client mouse messages.
Client2Screen (see page 147)	Corrected version of ClientToScreen function to support RTL.
ClientOrg (see page 147)	Corrected version of ClientOrigin function to support RTL.
Create (see page 147)	Creates and initializes a TBaseDesigner instance.
DesignState (see page 147)	Indicates whether component in design or loading state. At design time (in Delphi IDE) and during loading no activation occurs.
Destroy (see page 147)	Destroys an instance of TBaseDesigner.
DoObjectHint (see page 147)	Called to show hint for current design state.
DragDrop (see page 147)	OnDragDrop (see page 152) event dispatcher.
DragOver (see page 148)	OnDragOver (see page 152) event dispatcher.
IsRTL (see page 148)	Returns true when control is right-to-left.
KeyDown (see page 148)	Respond to key press events.
KeyPress (see page 149)	Respond to keyboard input.
KeyUp (see page 149)	Respond to released key.
Loaded (see page 149)	Initializes the component after the form file has been read into memory.
MouseDown (see page 150)	OnMouseDown (see page 154) event dispatcher.
MouseMove (see page 150)	OnMouseMove (see page 154) event dispatcher.
MouseUp (see page 150)	OnMouseUp (see page 154) event dispatcher.

◆ ProcessMessage (see page 151)	Translates messages of all managed by designer controls.
◆ ResetHint (see page 151)	Resets hint for another component
◆ Screen2Client (see page 151)	Corrected version of ScreenToClient function to support RTL.
◆ V SetActive (see page 151)	Set method for Active (see page 151) property.
◆ ShowHint (see page 151)	Indicates whether hints should be shown for controls in the Designer.

## IClassSelector Interface

IClassSelector Interface	Description
◆ ClsChanged (see page 511)	Called when selected in component palette component class was changed.
◆ ClsPalChanged (see page 512)	Called when component palette was changed.

## TzCustomFormDesigner Class

TzCustomFormDesigner Class	Description
◆ AddCompEditorMenu (see page 170)	Adds menu items to the popup Menu associated with the component editor of selected in designer component. To remove previously added component editor's menu items use ClearCompEditorMenu (see page 171) method.
◆ CanRedo (see page 170)	Indicates whether the designer contains undone changes that can be repeated.
◆ CanUndo (see page 171)	Indicates whether the designer contains changes that can be backed out.
◆ CheckAction (see page 171)	Determines whether design operation specified by the Action parameter can be executed.
◆ ClearCompEditorMenu (see page 171)	Removes component editor's menu items added by the AddCompEditorMenu (see page 170) method.
◆ ClearUndo (see page 171)	Clears the undo (see page 183) buffer so that no changes to the Target (see page 193) can be backed out.
◆ CloseTextEditor (see page 171)	Closes in-place editor.
◆ V DragDrop (see page 171)	OnDragDrop event dispatcher.
◆ V DragOver (see page 171)	OnDragOver event dispatcher.
◆ EditAction (see page 172)	Executes designer operation specified by the Action parameter.
◆ AlignSelected (see page 172)	Performs alignment operation on selected components.
◆ AlignToGrid (see page 172)	Aligns selected components to the closest grid point.
◆ BringToFront (see page 172)	Moves a selected component in front of all other components on the form. This is called changing the component's z-order.
◆ BuildLocalMenu (see page 172)	Creates default popup menu.
◆ CancelDrag (see page 173)	Cancels current drag&drop design operation.
◆ CanDelete (see page 173)	Check if component can be deleted.
◆ GetEditState (see page 173)	Returns set of possible designer operations.
◆ CanInsert (see page 173)	Check if component can be inserted.
◆ CanMove (see page 173)	Check if component can be moved.
◆ CanPaste (see page 173)	Checks clipboard for components saved to it.
◆ CanRename (see page 173)	Check if component can be renamed.
◆ CanResize (see page 174)	Check if component can be resized.
◆ CanSelect (see page 174)	Check if component can be selected.
◆ ClearSelection (see page 174)	Resets selection.
◆ CopySelection (see page 174)	
◆ V Create (see page 174)	Creates and initializes a TzCustomFormDesigner instance.
◆ CutSelection (see page 174)	Cuts selected components to clipboard.
◆ DeleteSelection (see page 175)	Deletes the selected component or components
◆ V Destroy (see page 175)	Destroys an instance of TzCustomFormDesigner.
◆ V DoObjectHint (see page 175)	Shows hint current design state.
◆ V ExecuteAction (see page 175)	Invokes an action with the component as its target.
◆ DragDraw (see page 175)	Provides drag-and-draw graphic operation.
◆ Edit (see page 175)	Displays the component editor for the specified component.
◆ EndDrag (see page 176)	Provides drag-and-draw graphic operation.
◆ FlipChildren (see page 176)	Allows to reverse the layout of components in the current form to a right-to-left mirror image.
◆ GetCompObj (see page 176)	Returns component
◆ GetComponent (see page 176)	Returns the component with the name passed as a parameter.
◆ GetComponentName (see page 176)	Returns the name of the component passed as its parameter.
◆ GetComponentNames (see page 177)	Executes a callback for every component that can be assigned a property of a specified type.

⇒ GetControlAt (see page 177)	Looks for control at specified point on the window prn.
⇒ GetMethodName (see page 177)	Returns the name of a specified event handler.
⇒ GetNewName (see page 177)	Returns new generated name for particular class of the component.
⇒ GetObjectName (see page 177)	Returns object name.
⇒ LoadFromFile (see page 177)	Load Root (see page 192) component from file. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.
⇒ LoadFromStream (see page 178)	Load Root (see page 192) component from stream. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.
⇒ GetRoot (see page 178)	Returns the current entity being edited by the form designer.
⇒ GetRootClassName (see page 178)	Returns the class name for the root component.
⇒ GetScriptEvent (see page 178)	Returns script procedure name assigned to the event.
⇒ GetScrollRanges (see page 178)	Returns the size of the logical designer window.
⇒ GetSelections (see page 178)	Fills a list with all selected components on the current root object.
⇒ Redo (see page 179)	Call Redo to repeat last undone operation.
⇒ GetShiftState (see page 179)	Returns the current state of the Shift, Alt, and Ctrl keys.
⇒ Intf_Notification (see page 179)	Allows the designer to respond when a notification (see page 183) is sent to the form.
⇒ IsComponentHidden (see page 179)	Indicates whether a component does not appear directly in the form designer.
⇒ IsLocked (see page 179)	Specifies whether component is locked and can not be edited in designer.
⇒ SaveToFile (see page 179)	Saves Root (see page 192) component to file FileName with events information. AsText specifies format of the file: text or binary.
⇒ IsDesignMsg (see page 180)	Determines when the designer should handle a Windows message.
⇒ SaveToStream (see page 180)	Save Root (see page 192) component to stream with events information. AsText specifies storage format: text or binary.
⇒ IsProtected (see page 180)	Indicates whether component is protected and can not be changed by designer.
⇒ IsRootSelected (see page 180)	Returns True if entire Root (see page 192) is selected
⇒ IsSourceReadOnly (see page 180)	Indicates whether the source file for the component being designed is read-only.
⇒ SelectedComponentsCount (see page 180)	Returns number of selected components excluding Root (see page 192) and non-component objects.
⇒ KeyDown (see page 180)	Respond to key press events.
⇒ KeyPress (see page 181)	Respond to keyboard input.
⇒ KeyUp (see page 181)	Respond to released key.
⇒ MethodExists (see page 181)	Indicates whether an event handler with a specified name already exists.
⇒ Modified (see page 181)	Notifies property and component editors when a change is made to a component.
⇒ ShowTabOrder (see page 182)	Shows tab order icons over children controls of the selected control. Click on the children controls changes their tab order. To exit "Show Tab Icons" mode click on any not child control or press ESCAPE key.
⇒ MouseDown (see page 182)	Generates an OnMouseDown event.
⇒ MouseMove (see page 182)	Generates an OnMouseMove event.
⇒ MouseUp (see page 182)	Generates an OnMouseUp event.
⇒ Navigate (see page 182)	Performs navigation between control using keyboard
⇒ UpdateAction (see page 182)	Updates an action component to reflect the current state of the component.
⇒ NoSelection (see page 183)	Deselects all components in the form designer.
⇒ Undo (see page 183)	Backs out last change in the undo buffer.
⇒ Notification (see page 183)	Allows the designer to respond when a notification is sent to the form.
⇒ NotifySelChanged (see page 183)	Notifies active designer about changing selection list
⇒ PaintControl (see page 184)	Called for each WM_PAINT message to perform specific painting over control.
⇒ PaintGrid (see page 184)	Paints the alignment grid on the form's canvas.
⇒ PasteSelection (see page 184)	Pastes the contents of the clipboard into the selected component or components.
⇒ ReadComp (see page 184)	Callback procedure for TReader.ReadComponents
⇒ RenameMethod (see page 184)	Renames an existing event handler.
⇒ Scale (see page 185)	Scale all controls using defined ratio
⇒ SelectAll (see page 185)	Selects all components.
⇒ SelectComponent (see page 185)	Replaces the current set of selected components by a single specified object.



SelectedComponent ( <a href="#">see page 185</a> )	Returns selected component if selection consist of single component; Root ( <a href="#">see page 192</a> ) component, if there is no selection; nil if selection consists of several components.
SelectionChanged ( <a href="#">see page 185</a> )	This method call after selection has been changed.
SelectObj ( <a href="#">see page 185</a> )	Selects object passed as Instance parameter.
SelectRect ( <a href="#">see page 185</a> )	Selects all controls on the Prn control.
SendToBack ( <a href="#">see page 186</a> )	Moves a selected component behind all other components on the form. This is called changing the component's z-order.
SetPasteName ( <a href="#">see page 186</a> )	Setting new name for pasting component
SetScriptEvent ( <a href="#">see page 186</a> )	Assigns script procedure with particular event of the Instance.
SetSelections ( <a href="#">see page 186</a> )	Changes the currently selected set of components.
ShowMethod ( <a href="#">see page 186</a> )	Activates the code editor with the input cursor in a specified event handler.
ShowPopupMenu ( <a href="#">see page 186</a> )	Show designer popup menu at the specified screen position.
SizeSelected ( <a href="#">see page 186</a> )	Perform size operation to selected components
StartDrag ( <a href="#">see page 187</a> )	Makes start settings before dragging operation
UniqueName ( <a href="#">see page 187</a> )	Generates a unique name from a specified base string.
UpdateComplcons ( <a href="#">see page 187</a> )	Updates positions of component icons.
ValidateMethod ( <a href="#">see page 187</a> )	Determines whether method MAddr of object ARoot is valid method, i.e. it may be assigned to the event.

### TBaseDesigner Properties

TBaseDesigner Properties	Description
Active ( <a href="#">see page 151</a> )	Switches target component between design and run-time modes
HintObject ( <a href="#">see page 152</a> )	Specifies object for which hint was activated.
ShowHints ( <a href="#">see page 152</a> )	Specifies showing of design hints.

### TzCustomFormDesigner Class

TzCustomFormDesigner Class	Description
AllowComponents ( <a href="#">see page 187</a> )	Specifies whether nonvisual components will be displayed in design mode
AutoAlign ( <a href="#">see page 187</a> )	Specifies using of align rulers.
BDSStyle ( <a href="#">see page 188</a> )	Specifies using of BDS style design environment.
CaptionFont ( <a href="#">see page 188</a> )	Controls the text attributes of non-visual components captions.
CloseDisactive ( <a href="#">see page 188</a> )	Specifies if TzCustomFormDesigner automatically deactivates when Target ( <a href="#">see page 193</a> ) Form ( <a href="#">see page 189</a> ) is to be closed.
ContainerWindow ( <a href="#">see page 188</a> )	Determines generic container for any type of Target ( <a href="#">see page 193</a> ) components
DesignSurface ( <a href="#">see page 188</a> )	Specifies design surface.
DisplayControlGrid ( <a href="#">see page 189</a> )	Specifies whether designer has to paint grid over window controls that can accept controls.
DisplayGrid ( <a href="#">see page 189</a> )	Determines whether dots are drawn on the Target ( <a href="#">see page 193</a> ) form.
FlatIcons ( <a href="#">see page 189</a> )	Determines whether the non-visual component icons has a 3D border or not.
DragParentLimit ( <a href="#">see page 189</a> )	Specifies whether drag mouse movement should be clipped by parent's client area.
Form ( <a href="#">see page 189</a> )	Provides access to designed form as TCustomForm type.
Events ( <a href="#">see page 189</a> )	Storage of assigned events.
GridStepX ( <a href="#">see page 190</a> )	Specifies grid step, in pixels, along X-axis
GridStepY ( <a href="#">see page 190</a> )	Specifies grid step, in pixels, along Y-axis
LockControls ( <a href="#">see page 190</a> )	Specifies if user can directly change size and position of controls by mouse.
LockPublished ( <a href="#">see page 190</a> )	Specifies if editing operation are forbidden by default
MultiSelect ( <a href="#">see page 190</a> )	Determines whether the user can select more than one control at a time.
Groups ( <a href="#">see page 190</a> )	Stores groups information.
GuidelinesStyle ( <a href="#">see page 191</a> )	Specifies guidelines options.
IgnoreReadErrors ( <a href="#">see page 191</a> )	Specifies whether read errors should be ignored when loading using LoadFromFile ( <a href="#">see page 177</a> ) and LoadFromStream ( <a href="#">see page 178</a> ) methods.
ReadOnly ( <a href="#">see page 191</a> )	Set Read Only mode.
StoreEvents ( <a href="#">see page 191</a> )	Specifies whether designer should process events storage.
TabOrderIcons ( <a href="#">see page 191</a> )	Properties of tab order icons. These controls are shown over children control when designer is in "Show Tab Order" mode.
TextEditMode ( <a href="#">see page 191</a> )	Sets in-place text editing mode of designer.
PopupMenu ( <a href="#">see page 191</a> )	Identifies the pop-up menu associated with the Root ( <a href="#">see page 192</a> ) control of the Designer.







UndoLimit (see page 192)	Specifies the number of changes that can be undone.
PopupMenuFilter (see page 192)	Specifies which categories of items may be used to form default popup menu.
UndoLoad (see page 192)	Indicates that designer is in Undo (see page 183) loading state, i.e. in reading previously saved form resource.
Root (see page 192)	Root component for TzCustomFormDesigner.
RootModified (see page 192)	Indicates whether the Root (see page 192) or its components are modified (see page 181).
SelCount (see page 193)	Indicates number of selected components.
Selected (see page 193)	Indicates whether a particular control is selected.
SelMarker (see page 193)	Selection markers manager.
ShowCaptions (see page 193)	Specifies whether non-visual component icons captions are visible.
SnapToGrid (see page 193)	Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines.
Target (see page 193)	Specifies object that is edited by the designer.

## TBaseDesigner Events







TBaseDesigner Events	Description
OnActiveChanged (see page 152)	Occurs when the Active (see page 151) property of the TzCustomFormDesigner changes
OnDragDrop (see page 152)	Occurs when the user drops an object being dragged.
OnDragOver (see page 152)	Occurs when the user drags an object over a control.
OnHandleControlMessage (see page 153)	Occurs on any message sent to managed controls.
OnKeyDown (see page 153)	Occurs only at design mode when user presses down any key.
OnKeyPress (see page 153)	Occurs only at design mode when key pressed.
OnKeyUp (see page 154)	Occurs only at design mode when user releases key that has been pressed.
OnMouseDown (see page 154)	Occurs only at design mode when user presses mouse button.
OnMouseMove (see page 154)	Occurs only at design mode when user moves mouse.
OnMouseUp (see page 154)	Occurs only at design mode when user releases mouse button.

## TzCustomFormDesigner Class











TzCustomFormDesigner Class	Description
OnCanEdit (see page 194)	Occurs to determine whether Edit (see page 175) method of component editor can be called.
OnDrawControl (see page 195)	Occurs when painting any control on the form. Use this event to draw over control.
OnExecuteAction (see page 195)	Occurs when ExecuteAction (see page 175) method is called. Use it to handle standard shared actions for currently active designer.
OnGetComponentLocked (see page 195)	Occurs to determine whether component is locked.
OnCanDelete (see page 195)	Occurs before deleting selected component.
OnCanInsert (see page 195)	Occurs before inserting new component.
OnGetObjectName (see page 195)	Occurs at the end of GetObjectName (see page 177) method to adjust resulting name.
OnCanMove (see page 196)	Occurs before moving selected component.
OnCanRename (see page 196)	Occurs before renaming component.
OnCanResize (see page 196)	Occurs before resizing selected component.
OnPopUndo (see page 196)	Occurs when restoring Target (see page 193) from undo (see page 183) buffer.
OnCanSelect (see page 196)	Occurs before selecting component.
OnPushUndo (see page 197)	Occurs when saving Target (see page 193) to undo (see page 183) buffer.
OnCreateComponent (see page 197)	Occurs before new component creation.
OnCreateFrame (see page 197)	Occurs when frame is to be inserted on the form.
OnSetNewName (see page 197)	Occurs when assigning name to newly inserted component (created or pasted).
OnCreateIcon (see page 197)	Occurs before creating icon for non-visual component.
OnCreateMethod (see page 198)	Occurs when new method name is input in object inspector.
OnFormClosed (see page 198)	Occurs immediately after hiding the Target (see page 193) form .
OnUpdateAction (see page 198)	Occurs when UpdateAction (see page 182) method is called. Use it to handle standard shared actions for currently active designer.
OnGetComponentHint (see page 198)	Occurs when the application is about to display the hint window for the particular component.
OnGetMethodNames (see page 198)	Occurs when method property editor requests designer for possible method names.

 OnGetScriptProc (see page 199)	Occurs when method property editor ask for script procedure name associated with a given property.
 OnNotification (see page 199)	Occurs when components are added or removed to/from Root (see page 192) object at design mode.
 OnRenameMethod (see page 200)	Occurs when name of method is changed in object inspector.
 OnSetScriptProc (see page 200)	Occurs when method property editor assigns script procedure to the event.
 OnShowMethod (see page 201)	Occurs when user double clicks on the procedure in the object inspector.
 OnValidateMethod (see page 201)	Occurs to validate method.






















## Legend








	Method
	protected
	virtual
	Property
	read only
	Event

## TBaseDesigner Events























TBaseDesigner Events	Description
 OnActiveChanged (see page 152)	Occurs when the Active (see page 151) property of the TzCustomFormDesigner changes
 OnDragDrop (see page 152)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 152)	Occurs when the user drags an object over a control.
 OnHandleControlMessage (see page 153)	Occurs on any message sent to managed controls.
 OnKeyDown (see page 153)	Occurs only at design mode when user presses down any key.
 OnKeyPress (see page 153)	Occurs only at design mode when key pressed.
 OnKeyUp (see page 154)	Occurs only at design mode when user releases key that has been pressed.
 OnMouseDown (see page 154)	Occurs only at design mode when user presses mouse button.
 OnMouseMove (see page 154)	Occurs only at design mode when user moves mouse.
 OnMouseUp (see page 154)	Occurs only at design mode when user releases mouse button.

## TzCustomFormDesigner Class



TzCustomFormDesigner Class	Description
 OnCanEdit (see page 194)	Occurs to determine whether Edit (see page 175) method of component editor can be called.
 OnDrawControl (see page 195)	Occurs when painting any control on the form. Use this event to draw over control.
 OnExecuteAction (see page 195)	Occurs when ExecuteAction (see page 175) method is called. Use it to handle standard shared actions for currently active designer.
 OnGetComponentLocked (see page 195)	Occurs to determine whether component is locked.
 OnCanDelete (see page 195)	Occurs before deleting selected component.
 OnCanInsert (see page 195)	Occurs before inserting new component.
 OnGetObjectName (see page 195)	Occurs at the end of GetObjectName (see page 177) method to adjust resulting name.
 OnCanMove (see page 196)	Occurs before moving selected component.
 OnCanRename (see page 196)	Occurs before renaming component.
 OnCanResize (see page 196)	Occurs before resizing selected component.
 OnPopUndo (see page 196)	Occurs when restoring Target (see page 193) from undo (see page 183) buffer.
 OnCanSelect (see page 196)	Occurs before selecting component.
 OnPushUndo (see page 197)	Occurs when saving Target (see page 193) to undo (see page 183) buffer.
 OnCreateComponent (see page 197)	Occurs before new component creation.
 OnCreateFrame (see page 197)	Occurs when frame is to be inserted on the form.
 OnSetNewName (see page 197)	Occurs when assigning name to newly inserted component (created or pasted).
 OnCreatelcon (see page 197)	Occurs before creating icon for non-visual component.
 OnCreateMethod (see page 198)	Occurs when new method name is input in object inspector.
 OnFormClosed (see page 198)	Occurs immediately after hiding the Target (see page 193) form .
 OnUpdateAction (see page 198)	Occurs when UpdateAction (see page 182) method is called. Use it to handle standard shared actions for currently active designer.
 OnGetComponentHint (see page 198)	Occurs when the application is about to display the hint window for the particular component.

 OnGetMethodNames (see page 198)	Occurs when method property editor requests designer for possible method names.
 OnGetScriptProc (see page 199)	Occurs when method property editor ask for script procedure name associated with a given property.
 OnNotification (see page 199)	Occurs when components are added or removed to/from Root (see page 192) object at design mode.
 OnRenameMethod (see page 200)	Occurs when name of method is changed in object inspector.
 OnSetScriptProc (see page 200)	Occurs when method property editor assigns script procedure to the event.
 OnShowMethod (see page 201)	Occurs when user double clicks on the procedure in the object inspector.
 OnValidateMethod (see page 201)	Occurs to validate method.













## TBaseDesigner Methods

TBaseDesigner Methods	Description
 CanProcessNCMessages (see page 146)	Called to check processing of non client mouse messages.
 Client2Screen (see page 147)	Corrected version of ClientToScreen function to support RTL.
 ClientOrg (see page 147)	Corrected version of ClientOrigin function to support RTL.
 Create (see page 147)	Creates and initializes a TBaseDesigner instance.
 DesignState (see page 147)	Indicates whether component in design or loading state. At design time (in Delphi IDE) and during loading no activation occurs.
 Destroy (see page 147)	Destroys an instance of TBaseDesigner.
 DoObjectHint (see page 147)	Called to show hint for current design state.
 DragDrop (see page 147)	OnDragDrop (see page 152) event dispatcher.
 DragOver (see page 148)	OnDragOver (see page 152) event dispatcher.
 IsRTL (see page 148)	Returns true when control is right-to-left.
 KeyDown (see page 148)	Respond to key press events.
 KeyPress (see page 149)	Respond to keyboard input.
 KeyUp (see page 149)	Respond to released key.
 Loaded (see page 149)	Initializes the component after the form file has been read into memory.
 MouseDown (see page 150)	OnMouseDown (see page 154) event dispatcher.
 MouseMove (see page 150)	OnMouseMove (see page 154) event dispatcher.
 MouseUp (see page 150)	OnMouseUp (see page 154) event dispatcher.
 ProcessMessage (see page 151)	Translates messages of all managed by designer controls.
 ResetHint (see page 151)	Resets hint for another component
 Screen2Client (see page 151)	Corrected version of ScreenToClient function to support RTL.
 SetActive (see page 151)	Set method for Active (see page 151) property.
 ShowHint (see page 151)	Indicates whether hints should be shown for controls in the Designer.

## IClassSelector Interface

IClassSelector Interface	Description
 ClsChanged (see page 511)	Called when selected in component palette component class was changed.
 ClsPalChanged (see page 512)	Called when component palette was changed.

## TzCustomFormDesigner Class

TzCustomFormDesigner Class	Description
 AddCompEditorMenu (see page 170)	Adds menu items to the popup Menu associated with the component editor of selected in designer component. To remove previously added component editor's menu items use ClearCompEditorMenu (see page 171) method.
 CanRedo (see page 170)	Indicates whether the designer contains undone changes that can be repeated.
 CanUndo (see page 171)	Indicates whether the designer contains changes that can be backed out.
 CheckAction (see page 171)	Determines whether design operation specified by the Action parameter can be executed.
 ClearCompEditorMenu (see page 171)	Removes component editor's menu items added by the AddCompEditorMenu (see page 170) method.
 ClearUndo (see page 171)	Clears the undo (see page 183) buffer so that no changes to the Target (see page 193) can be backed out.
 CloseTextEditor (see page 171)	Closes in-place editor.
 DragDrop (see page 171)	OnDragDrop event dispatcher.
 DragOver (see page 171)	OnDragOver event dispatcher.
 EditAction (see page 172)	Executes designer operation specified by the Action parameter.
 AlignSelected (see page 172)	Performs alignment operation on selected components.
 AlignToGrid (see page 172)	Aligns selected components to the closest grid point.

◆ BringToFront (see page 172)	Moves a selected component in front of all other components on the form. This is called changing the component's z-order.
◆ BuildLocalMenu (see page 172)	Creates default popup menu.
◆ CancelDrag (see page 173)	Cancels current drag&drop design operation.
◆ CanDelete (see page 173)	Check if component can be deleted.
◆ GetEditState (see page 173)	Returns set of possible designer operations.
◆ CanInsert (see page 173)	Check if component can be inserted.
◆ CanMove (see page 173)	Check if component can be moved.
◆ CanPaste (see page 173)	Checks clipboard for components saved to it.
◆ CanRename (see page 173)	Check if component can be renamed.
◆ CanResize (see page 174)	Check if component can be resized.
◆ CanSelect (see page 174)	Check if component can be selected.
◆ ClearSelection (see page 174)	Resets selection.
◆ CopySelection (see page 174)	
◆ Create (see page 174)	Creates and initializes a TzCustomFormDesigner instance.
◆ CutSelection (see page 174)	Cuts selected components to clipboard.
◆ DeleteSelection (see page 175)	Deletes the selected component or components
◆ Destroy (see page 175)	Destroys an instance of TzCustomFormDesigner.
◆ DoObjectHint (see page 175)	Shows hint current design state.
◆ ExecuteAction (see page 175)	Invokes an action with the component as its target.
◆ DragDraw (see page 175)	Provides drag-and-draw graphic operation.
◆ Edit (see page 175)	Displays the component editor for the specified component.
◆ EndDrag (see page 176)	Provides drag-and-draw graphic operation.
◆ FlipChildren (see page 176)	Allows to reverse the layout of components in the current form to a right-to-left mirror image.
◆ GetCompObj (see page 176)	Returns component
◆ GetComponent (see page 176)	Returns the component with the name passed as a parameter.
◆ GetComponentName (see page 176)	Returns the name of the component passed as its parameter.
◆ GetComponentNames (see page 177)	Executes a callback for every component that can be assigned a property of a specified type.
◆ GetControlAt (see page 177)	Looks for control at specified point on the window prn.
◆ GetMethodName (see page 177)	Returns the name of a specified event handler.
◆ GetNewName (see page 177)	Returns new generated name for particular class of the component.
◆ GetObjectName (see page 177)	Returns object name.
◆ LoadFromFile (see page 177)	Load Root (see page 192) component from file. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.
◆ LoadFromStream (see page 178)	Load Root (see page 192) component from stream. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.
◆ GetRoot (see page 178)	Returns the current entity being edited by the form designer.
◆ GetRootClassName (see page 178)	Returns the class name for the root component.
◆ GetScriptEvent (see page 178)	Returns script procedure name assigned to the event.
◆ GetScrollRanges (see page 178)	Returns the size of the logical designer window.
◆ GetSelections (see page 178)	Fills a list with all selected components on the current root object.
◆ Redo (see page 179)	Call Redo to repeat last undone operation.
◆ GetShiftState (see page 179)	Returns the current state of the Shift, Alt, and Ctrl keys.
◆ Intf_Notification (see page 179)	Allows the designer to respond when a notification (see page 183) is sent to the form.
◆ IsComponentHidden (see page 179)	Indicates whether a component does not appear directly in the form designer.
◆ IsLocked (see page 179)	Specifies whether component is locked and can not be edited in designer.
◆ SaveToFile (see page 179)	Saves Root (see page 192) component to file FileName with events information. AsText specifies format of the file: text or binary.
◆ IsDesignMsg (see page 180)	Determines when the designer should handle a Windows message.
◆ SaveToStream (see page 180)	Save Root (see page 192) component to stream with events information. AsText specifies storage format: text or binary.
◆ IsProtected (see page 180)	Indicates whether component is protected and can not be changed by designer.
◆ IsRootSelected (see page 180)	Returns True if entire Root (see page 192) is selected
◆ IsSourceReadOnly (see page 180)	Indicates whether the source file for the component being designed is read-only.

SelectedComponentsCount (see page 180)	Returns number of selected components excluding Root (see page 192) and non-component objects.
KeyDown (see page 180)	Respond to key press events.
KeyPress (see page 181)	Respond to keyboard input.
KeyUp (see page 181)	Respond to released key.
MethodExists (see page 181)	Indicates whether an event handler with a specified name already exists.
Modified (see page 181)	Notifies property and component editors when a change is made to a component.
ShowTabOrder (see page 182)	Shows tab order icons over children controls of the selected control. Click on the children controls changes their tab order. To exit "Show Tab Icons" mode click on any not child control or press ESCAPE key.
MouseDown (see page 182)	Generates an OnMouseDown event.
MouseMove (see page 182)	Generates an OnMouseMove event.
MouseUp (see page 182)	Generates an OnMouseUp event.
Navigate (see page 182)	Performs navigation between control using keyboard
UpdateAction (see page 182)	Updates an action component to reflect the current state of the component.
NoSelection (see page 183)	Deselects all components in the form designer.
Undo (see page 183)	Backs out last change in the undo buffer.
Notification (see page 183)	Allows the designer to respond when a notification is sent to the form.
NotifySelChanged (see page 183)	Notifies active designer about changing selection list
PaintControl (see page 184)	Called for each WM_PAINT message to perform specific painting over control.
PaintGrid (see page 184)	Paints the alignment grid on the form's canvas.
PasteSelection (see page 184)	Pastes the contents of the clipboard into the selected component or components.
ReadComp (see page 184)	Callback procedure for TReader.ReadComponents
RenameMethod (see page 184)	Renames an existing event handler.
Scale (see page 185)	Scale all controls using defined ratio
SelectAll (see page 185)	Selects all components.
SelectComponent (see page 185)	Replaces the current set of selected components by a single specified object.
SelectedComponent (see page 185)	Returns selected component if selection consist of single component; Root (see page 192) component, if there is no selection; nil if selection consists of several components.
SelectionChanged (see page 185)	This method call after selection has been changed.
SelectObj (see page 185)	Selects object passed as Instance parameter.
SelectRect (see page 185)	Selects all controls on the Prn control.
SendToBack (see page 186)	Moves a selected component behind all other components on the form. This is called changing the component's z-order.
SetPasteName (see page 186)	Setting new name for pasting component
SetScriptEvent (see page 186)	Assigns script procedure with particular event of the Instance.
SetSelections (see page 186)	Changes the currently selected set of components.
ShowMethod (see page 186)	Activates the code editor with the input cursor in a specified event handler.
ShowPopupMenu (see page 186)	Show designer popup menu at the specified screen position.
SizeSelected (see page 186)	Perform size operation to selected components
StartDrag (see page 187)	Makes start settings before dragging operation
UniqueName (see page 187)	Generates a unique name from a specified base string.
UpdateComplcons (see page 187)	Updates positions of component icons.
ValidateMethod (see page 187)	Determines whether method MAddr of object ARoot is valid method, i.e. it may be assigned to the event.

### TBaseDesigner Properties

TBaseDesigner Properties	Description
Active (see page 151)	Switches target component between design and run-time modes
HintObject (see page 152)	Specifies object for which hint was activated.
ShowHints (see page 152)	Specifies showing of design hints.

### TzCustomFormDesigner Class

TzCustomFormDesigner Class	Description
AllowComponents (see page 187)	Specifies whether nonvisual components will be displayed in design mode
AutoAlign (see page 187)	Specifies using of align rulers.
BDSStyle (see page 188)	Specifies using of BDS style design environment.
CaptionFont (see page 188)	Controls the text attributes of non-visual components captions.



 CloseDisactive ( <a href="#">see page 188</a> )	Specifies if TzCustomFormDesigner automatically deactivates when Target ( <a href="#">see page 193</a> ) Form ( <a href="#">see page 189</a> ) is to be closed.
 ContainerWindow ( <a href="#">see page 188</a> )	Determines generic container for any type of Target ( <a href="#">see page 193</a> ) components
 DesignSurface ( <a href="#">see page 188</a> )	Specifies design surface.
 DisplayControlGrid ( <a href="#">see page 189</a> )	Specifies whether designer has to paint grid over window controls that can accept controls.
 DisplayGrid ( <a href="#">see page 189</a> )	Determines whether dots are drawn on the Target ( <a href="#">see page 193</a> ) form.
 FlatIcons ( <a href="#">see page 189</a> )	Determines whether the non-visual component icons has a 3D border or not.
 DragParentLimit ( <a href="#">see page 189</a> )	Specifies whether drag mouse movement should be clipped by parent's client area.
 Form ( <a href="#">see page 189</a> )	Provides access to designed form as TCustomForm type.
 Events ( <a href="#">see page 189</a> )	Storage of assigned events.
 GridStepX ( <a href="#">see page 190</a> )	Specifies grid step, in pixels, along X-axis
 GridStepY ( <a href="#">see page 190</a> )	Specifies grid step, in pixels, along Y-axis
 LockControls ( <a href="#">see page 190</a> )	Specifies if user can directly change size and position of controls by mouse.
 LockPublished ( <a href="#">see page 190</a> )	Specifies if editing operation are forbidden by default
 MultiSelect ( <a href="#">see page 190</a> )	Determines whether the user can select more than one control at a time.
 R Groups ( <a href="#">see page 190</a> )	Stores groups information.
 GuidelinesStyle ( <a href="#">see page 191</a> )	Specifies guidelines options.
 IgnoreReadErrors ( <a href="#">see page 191</a> )	Specifies whether read errors should be ignored when loading using LoadFromFile ( <a href="#">see page 177</a> ) and LoadFromStream ( <a href="#">see page 178</a> ) methods.
 ReadOnly ( <a href="#">see page 191</a> )	Set Read Only mode.
 StoreEvents ( <a href="#">see page 191</a> )	Specifies whether designer should process events storage.
 TabOrderIcons ( <a href="#">see page 191</a> )	Properties of tab order icons. These controls are shown over children control when designer is in "Show Tab Order" mode.
 TextEditMode ( <a href="#">see page 191</a> )	Sets in-place text editing mode of designer.
 PopupMenu ( <a href="#">see page 191</a> )	Identifies the pop-up menu associated with the Root ( <a href="#">see page 192</a> ) control of the Designer.
 UndoLimit ( <a href="#">see page 192</a> )	Specifies the number of changes that can be undone.
 PopupMenuFilter ( <a href="#">see page 192</a> )	Specifies which categories of items may be used to form default popup menu.
 R UndoLoad ( <a href="#">see page 192</a> )	Indicates that designer is in Undo ( <a href="#">see page 183</a> ) loading state, i.e. in reading previously saved form resource.
 Root ( <a href="#">see page 192</a> )	Root component for TzCustomFormDesigner.
 RootModified ( <a href="#">see page 192</a> )	Indicates whether the Root ( <a href="#">see page 192</a> ) or its components are modified ( <a href="#">see page 181</a> ).
 R SelCount ( <a href="#">see page 193</a> )	Indicates number of selected components.
 R Selected ( <a href="#">see page 193</a> )	Indicates whether a particular control is selected.
 SelMarker ( <a href="#">see page 193</a> )	Selection markers manager.
 ShowCaptions ( <a href="#">see page 193</a> )	Specifies whether non-visual component icons captions are visible.
 SnapToGrid ( <a href="#">see page 193</a> )	Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines.
 Target ( <a href="#">see page 193</a> )	Specifies object that is edited by the designer.

### 1.15.1.3.1 TzCustomFormDesigner Methods

#### 1.15.1.3.1.1 TzCustomFormDesigner.AddCompEditorMenu Method

Adds menu items to the popup Menu associated with the component editor of selected in designer component.

To remove previously added component editor's menu items use ClearCompEditorMenu ([see page 171](#)) method.

**procedure** AddCompEditorMenu(Menu: TPopupMenu);

#### 1.15.1.3.1.2 TzCustomFormDesigner.CanRedo Method

Indicates whether the designer contains undone changes that can be repeated.

**function** CanRedo: Boolean;

#### Description

Read CanRedo to determine whether the user has made undo ([see page 183](#)) operations to the Target ([see page 193](#))

that can be undone by calling the Redo (see page 179) method. This is useful when enabling or disabling menu items that correspond to these commands.

#### 1.15.1.3.1.3 TzCustomFormDesigner.CanUndo Method

Indicates whether the designer contains changes that can be backed out.

```
function CanUndo: Boolean;
```

##### Description

Read CanUndo to determine whether the user has made any changes to the Target (see page 193) of the designer that can be undone by calling the Undo (see page 183) method. This is useful when enabling or disabling menu items that correspond to these commands.

#### 1.15.1.3.1.4 TzCustomFormDesigner.CheckAction Method

Determines whether design operation specified by the Action parameter can be executed.

```
function CheckAction(Action: TEditStates): Boolean;
```

#### 1.15.1.3.1.5 TzCustomFormDesigner.ClearCompEditorMenu Method

Removes component editor's menu items added by the AddCompEditorMenu (see page 170) method.

```
procedure ClearCompEditorMenu(Menu: TPopupMenu);
```

#### 1.15.1.3.1.6 TzCustomFormDesigner.ClearUndo Method

Clears the undo (see page 183) buffer so that no changes to the Target (see page 193) can be backed out.

```
procedure ClearUndo;
```

##### Description

Use ClearUndo to commit changes Target (see page 193) resource. After calling ClearUndo, the CanUndo (see page 171) property is false and the Undo (see page 183) method does nothing.

#### 1.15.1.3.1.7 TzCustomFormDesigner.CloseTextEditor Method

Closes in-place editor.

```
procedure CloseTextEditor(Accept: Boolean);
```

##### Description

If Accept is True, edited text will be saved to control, otherwise all not saved changes will be canceled.

#### 1.15.1.3.1.8 TzCustomFormDesigner.DragDrop Method

OnDragDrop event dispatcher.

```
procedure DragDrop(Sender: TObject; Source: TObject; X: Integer; Y: Integer); override;
```

##### Description

Override DragDrop to add additional code that executes before the OnDragDrop event handler is called.

The Source parameter is the object that was dropped onto the control Sender. The X and Y parameters are the mouse coordinates where the object was dropped.

#### 1.15.1.3.1.9 TzCustomFormDesigner.DragOver Method

OnDragOver event dispatcher.



```
function DragOver(Sender: TObject; Source: TObject; X: Integer; Y: Integer; State: TDragState): Boolean; override;
```

**Description**

Override DragOver to add additional code that executes before the OnDragOver event handler is called.

DragOver sets the Accept parameter to true to indicate that the user can drop the dragged object on the control. It sets Accept to false to indicate that the user cannot drop the dragged object on the control.

The Source parameter is the object being dragged. The State parameter indicates how the dragged object is moving in relation to the control. X and Y indicate the current position of the mouse.

**1.15.1.3.1.10 TzCustomFormDesigner.EditAction Method**

Executes designer operation specified by the Action parameter.

```
function EditAction(Action: TEditAction): Boolean;
```

**1.15.1.3.1.11 TzCustomFormDesigner.AlignSelected Method**

Performs alignment operation on selected components.

```
procedure AlignSelected(Horz: TCompAlign; Vert: TCompAlign);
```

**Description**

AlignSelected is called automatically if user set new values in TAlignmentDlg form

Call this property programmatically to set new alignment for the selected components

Horz, Vert parameters are values for horizontal and vertical alignment correspondingly.

**1.15.1.3.1.12 TzCustomFormDesigner.AlignToGrid Method**

Aligns selected components to the closest grid point.

```
procedure AlignToGrid;
```

**Description**

Execute inner designer's command dsnAlignToGrid.

Call AlignToGrid programmatically to align selected components so that none of them is placed "in between" gridlines.

**1.15.1.3.1.13 TzCustomFormDesigner.BringToFront Method**

Moves a selected component in front of all other components on the form. This is called changing the component's z-order.

```
procedure BringToFront;
```

**1.15.1.3.1.14 TzCustomFormDesigner.BuildLocalMenu Method**

Creates default popup menu.

```
function BuildLocalMenu(Base: TPopupMenu; Filter: TLocalMenuFilters): TPopupMenu;
```

**Description**

### 1.15.1.3.1.15 TzCustomFormDesigner.CancelDrag Method

Cancels current drag&drop design operation.

```
procedure CancelDrag;
```

#### Description

This method is called when users press Esc key during dragging operation.

### 1.15.1.3.1.16 TzCustomFormDesigner.CanDelete Method

Check if component can be deleted.

```
function CanDelete(Component: TComponent): Boolean; dynamic;
```

#### Description

CanDelete checks if component is not protected from deleting and than calls OnCanDelete (see page 195) event handler.

### 1.15.1.3.1.17 TzCustomFormDesigner.GetEditState Method

Returns set of possible designer operations.

```
function GetEditState: TEditState;
```

### 1.15.1.3.1.18 TzCustomFormDesigner.CanInsert Method

Check if component can be inserted.

```
function CanInsert(Component: TComponent): Boolean; dynamic;
```

#### Description

CanInsert simply calls OnCanInsert (see page 195) event handler where user can define whether a component is allowed to insert.

### 1.15.1.3.1.19 TzCustomFormDesigner.CanMove Method

Check if component can be moved.

```
function CanMove(Component: TComponent): Boolean; dynamic;
```

#### Description

CanMove calls OnCanMove (see page 196) event handler where user can define whether a component is allowed to move.

### 1.15.1.3.1.20 TzCustomFormDesigner.CanPaste Method

Checks clipboard for components saved to it.

```
function CanPaste: Boolean;
```

#### Description

Delphi IDE and EControl Form (see page 189) Designer Pro save components to clipboard in 'Delphi Components' clipboard format. Therefore they are compatible and you may copy component from/to Delphi IDE and EControl Form (see page 189) Designer Pro.

### 1.15.1.3.1.21 TzCustomFormDesigner.CanRename Method

Check if component can be renamed.

```
function CanRename(Component: TComponent; const NewName: string): Boolean; dynamic;
```

**Description**

CanMove (see page 173) calls OnCanRename (see page 196) event handler where user can define whether a component is allowed to rename.

**1.15.1.3.1.22 TzCustomFormDesigner.CanResize Method**

Check if component can be resized.

```
function CanResize(Component: TComponent): Boolean; dynamic;
```

**Description**

CanMove (see page 173) calls OnCanResize (see page 196) event handler where user can define whether a component is allowed to resize.

**1.15.1.3.1.23 TzCustomFormDesigner.CanSelect Method**

Check if component can be selected.

```
function CanSelect(Component: TComponent): Boolean; dynamic;
```

**Description**

CanMove (see page 173) calls OnCanSelect (see page 196) event handler where user can define whether a component is allowed to select.

**1.15.1.3.1.24 TzCustomFormDesigner.ClearSelection Method**

Resets selection.

```
procedure ClearSelection;
```

**Description**

This method set selection to nil, i.e. after calling this method Root (see page 192) is selected in object inspector and there are no selected controls.

**1.15.1.3.1.25 TzCustomFormDesigner.CopySelection Method**

```
procedure CopySelection;
```

**1.15.1.3.1.26 TzCustomFormDesigner.Create Constructor**

Creates and initializes a TzCustomFormDesigner instance.

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

**Description**

Use Create to programmatically instantiate a TzCustomFormDesigner component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

**1.15.1.3.1.27 TzCustomFormDesigner.CutSelection Method**

Cuts selected components to clipboard.

```
procedure CutSelection;
```

**Description**

Delphi IDE and EControl Form (see page 189) Designer Pro save components to clipboard in 'Delphi Components' clipboard format. Therefore they are compatible and you may copy component from/to Delphi IDE and EControl Form (see page 189) Designer Pro.

### 1.15.1.3.1.28 TzCustomFormDesigner.DeleteSelection Method

Deletes the selected component or components

```
procedure DeleteSelection(ADoAll: Boolean = False);
```

#### Description

Call DeleteSelection to remove the selected components in the designer and free their memory.

Use OnCanDelete (see page 195) event handler to define which components can not be deleted.

### 1.15.1.3.1.29 TzCustomFormDesigner.Destroy Destructor

Destroys an instance of TzCustomFormDesigner.

```
destructor Destroy; override;
```

#### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

### 1.15.1.3.1.30 TzCustomFormDesigner.DoObjectHint Method

Shows hint current design state.

```
procedure DoObjectHint; override;
```

#### Description

Used internally.

### 1.15.1.3.1.31 TzCustomFormDesigner.ExecuteAction Method

Invokes an action with the component as its target.

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

#### Description

When the user invokes an action, VCL makes a series of calls to respond to that action. First, it generates an OnExecute event of the action list that contains the action. If the action list does not handle the OnExecute event, then the action is routed to the Application object's ExecuteAction method, which invokes the OnActionExecute event handler. If the OnActionExecute event handler does not handle the action, then it is routed to the action's OnExecute event handler. If that does not handle the action, the active control's ExecuteAction method is called.

The Action parameter specifies the action that was invoked. ExecuteAction returns true if the action was successfully dispatched, and false if the component could not handle the action. If ExecuteAction returns false for the active control, VCL calls the active form's ExecuteAction method. If this returns false, VCL tries all active controls in the form. If these all return false, VCL repeats the process with the main form, if that is different from the active form.

### 1.15.1.3.1.32 TzCustomFormDesigner.DragDraw Method

Provides drag-and-draw graphic operation.

```
procedure DragDraw;
```

#### Description

DragDraw is used internally in operation for selecting and dragging components on the designed form.

### 1.15.1.3.1.33 TzCustomFormDesigner.Edit Method

Displays the component editor for the specified component.

```
procedure Edit(const Component: IComponent);
```

**Description**

This method looks for component editor for give component and calls its method Edit. This method is called when user double clicks on the component.

**1.15.1.3.1.34 TzCustomFormDesigner.EndDrag Method**

Provides drag-and-draw graphic operation.

```
procedure EndDrag(Shift: TShiftState);
```

**Description**

EndDraw is used internally in operation for selecting and dragging components on the designed form.

**1.15.1.3.1.35 TzCustomFormDesigner.FlipChildren Method**

Allows to reverse the layout of components in the current form to a right-to-left mirror image.

```
procedure FlipChildren(All: Boolean);
```

**Description**

This lets developers quickly change a form created for an audience that reads left to right so that it appears natural in environments where users read from right to left.

**1.15.1.3.1.36 TzCustomFormDesigner.GetCompObj Method**

Returns component

```
function GetCompObj(AControl: TControl): TComponent;
```

**Description****1.15.1.3.1.37 TzCustomFormDesigner.GetComponent Method**

Returns the component with the name passed as a parameter.

```
function GetComponent(const Name: string): TComponent;
```

**Description**

Call GetComponent to access a component given its name. If the component is not in the current root object, the Name parameter should include the name of the entity in which it resides. For example, to obtain a reference to a component in a data module named 'DataModule2', use a line such as

```
TheComponent := Designer.GetComponent('DataModule2.Button1');
```

**1.15.1.3.1.38 TzCustomFormDesigner.GetComponentName Method**

Returns the name of the component passed as its parameter.

```
function GetComponentName(Component: TComponent): string;
```

**Description**

Call GetComponentName to obtain the name of a component. This is the inverse of GetComponent (see page 176).

**Note:** If the component is in the current root object, GetComponentName returns the Name property of the component itself. If the component is in another entity, GetComponentName qualifies the component name with the name of the object in which it resides.

### 1.15.1.3.1.39 TzCustomFormDesigner.GetComponentNames Method

Executes a callback for every component that can be assigned a property of a specified type.

```
procedure GetComponentNames(TypeData: PTypeData; Proc: TGetStrProc);
```

#### Description

Use GetComponentNames to call the procedure specified by the Proc parameter for every component that can be assigned a property that matches the TypeData parameter. For each component, Proc is called with its S parameter set to the name of the component. This parameter can be used to obtain a reference to the component by calling the GetComponent (see page 176) method.

**Note:** GetComponentNames calls Proc for components in global components that can be defined using TDesignerEvents.OnGetGlobalComponents event handler.

### 1.15.1.3.1.40 TzCustomFormDesigner.GetControlAt Method

Looks for control at specified point on the window prn.

```
function GetControlAt(prn: TWinControl; p: TPoint): TControl;
```

#### Description

Lets find Control under specified mouse point.

The Prn parameter is a Parent control for that specified control is looking for.

The P parameter is a mouse point.

### 1.15.1.3.1.41 TzCustomFormDesigner.GetMethodName Method

Returns the name of a specified event handler.

```
function GetMethodName(const Method: TMethod): string;
```

#### Description

Call GetMethodName to obtain the name of an event handler given a pointer to it.

### 1.15.1.3.1.42 TzCustomFormDesigner.GetNewName Method

Returns new generated name for particular class of the component.

```
function GetNewName(AClass: TClass): string;
```

#### Description

### 1.15.1.3.1.43 TzCustomFormDesigner.GetObjectName Method

Returns object name.

```
function GetObjectName(Instance: TPersistent): string;
```

### 1.15.1.3.1.44 TzCustomFormDesigner.LoadFromFile Method

Load Root (see page 192) component from file. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.

```
procedure LoadFromFile(const FileName: string);
```

#### 1.15.1.3.1.45 TzCustomFormDesigner.LoadFromStream Method

Load Root (see page 192) component from stream. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.

```
procedure LoadFromStream(Stream: TStream);
```

#### 1.15.1.3.1.46 TzCustomFormDesigner.GetRoot Method

Returns the current entity being edited by the form designer.

```
function GetRoot: TComponent;
```

##### Description

GetRoot is the read implementation of the Root (see page 192) property. It returns the object on which the form designer is working.

Root (see page 192) component is equal to Target (see page 193) property.

#### 1.15.1.3.1.47 TzCustomFormDesigner.GetRootClassName Method

Returns the class name for the root component.

```
function GetRootClassName: string;
```

##### Description

Call GetRootClassName to obtain the class name of the component specified by the Root (see page 192) property.

Instead of Delphi this method returns real class name of edited object, because no new class is created for edited object at runtime.

#### 1.15.1.3.1.48 TzCustomFormDesigner.GetScriptEvent Method

Returns script procedure name assigned to the event.

```
function GetScriptEvent(Instance: TObject; pInfo: PPropInfo): string;
```

##### Description

This method calls OnGetScriptProc (see page 199) event handler to get script procedure name.

#### 1.15.1.3.1.49 TzCustomFormDesigner.GetScrollRanges Method

Returns the size of the logical designer window.

```
function GetScrollRanges(const ScrollPosition: TPoint): TPoint;
```

##### Description

Call GetScrollRanges to determine the farthest point to which the designer can scroll. The ScrollPosition parameter specifies a point that must be included in the scroll range. GetScrollRanges returns the coordinates that minimally contain all components on the design surface plus the point specified by ScrollPosition.

#### 1.15.1.3.1.50 TzCustomFormDesigner.GetSelections Method

Fills a list with all selected components on the current root object.

```
procedure GetSelections(const List: IDesignerSelections);
```

##### Description

Use GetSelections to access every component currently selected on the root object returned by the GetRoot (see page 178) method.

Pass a component that supports the `IDesignerSelections` interface as the `List` parameter. `GetSelections` fills this component with references to the selected objects.

**Note:** `GetSelections` can return persistent objects that are not components, as well as components.

#### 1.15.1.3.1.51 `TzCustomFormDesigner.Redo` Method

Call `Redo` to repeat last undone operation.

```
procedure Redo;
```

##### Description

To determine whether `Redo` operation is possible, call `CanRedo` (see page 170) method.

#### 1.15.1.3.1.52 `TzCustomFormDesigner.GetShiftState` Method

Returns the current state of the `Shift`, `Alt`, and `Ctrl` keys.

```
function GetShiftState: TShiftState;
```

##### Description

Call `GetShiftState` to ascertain whether the `Shift`, `Alt`, and `Ctrl` keys are currently pressed. `GetShiftState` returns a `TShiftState` value that indicates the status of these keys.

#### 1.15.1.3.1.53 `TzCustomFormDesigner.Intf_Notification` Method

Allows the designer to respond when a notification (see page 183) is sent to the form.

```
procedure Intf_Notification(AnObject: TPersistent; Operation: TOperation);
```

##### Description

When a form receives a notification (see page 183), it calls the `Notification` (see page 183) method of the designer, allowing the designer to respond to all the notifications the form receives.

It is implementation of `IDesignerNotify.Notification` (see page 183).

```
procedure IDesignerNotify.Notification = Intf_Notification;
```

#### 1.15.1.3.1.54 `TzCustomFormDesigner.IsComponentHidden` Method

Indicates whether a component does not appear directly in the form designer.

```
function IsComponentHidden(Component: TComponent): Boolean;
```

##### Description

Call `IsComponentHidden` to determine whether the component specified by the `Component` parameter appears directly in the form designer. When `IsComponentHidden` returns true, the component does not appear as an icon or control. For example, menu items and field components are hidden components, and can only be accessed through a parent component's editor (the menu designer or fields editor). Hidden components are registered using the `RegisterNolcon` procedure.

#### 1.15.1.3.1.55 `TzCustomFormDesigner.IsLocked` Method

Specifies whether component is locked and can not be edited in designer.

```
function IsLocked(Component: TComponent): Boolean; dynamic;
```

#### 1.15.1.3.1.56 `TzCustomFormDesigner.SaveToFile` Method

Saves `Root` (see page 192) component to file `FileName` with events information. `AsText` specifies format of the file: text or binary.



```
procedure SaveToFile(const FileName: string; AsText: Boolean = True);
```

#### 1.15.1.3.1.57 TzCustomFormDesigner.IsDesignMsg Method

Determines when the designer should handle a Windows message.

```
function IsDesignMsg(Sender: TControl; var Message: TMessage): Boolean;
```

##### Description

IsDesignMsg is called for each message sent to a component in the designer. This method returns true if the message is a design message, meaning one the designer should handle for the component.

#### 1.15.1.3.1.58 TzCustomFormDesigner.SaveToStream Method

Save Root (see page 192) component to stream with events information. AsText specifies storage format: text or binary.

```
procedure SaveToStream(Stream: TStream; AsText: Boolean = True);
```

#### 1.15.1.3.1.59 TzCustomFormDesigner.IsProtected Method

Indicates whether component is protected and can not be changed by designer.

```
function IsProtected(Component: TComponent): Boolean; dynamic;
```

##### Description

#### 1.15.1.3.1.60 TzCustomFormDesigner.IsRootSelected Method

Returns True if entire Root (see page 192) is selected

```
function IsRootSelected: Boolean;
```

##### Description

Root (see page 192) is selected when there is no selection. I.e. to select root component you should call NoSelection (see page 183) method.

#### 1.15.1.3.1.61 TzCustomFormDesigner.IsSourceReadOnly Method

Indicates whether the source file for the component being designed is read-only.

```
function IsSourceReadOnly: Boolean;
```

##### Description

Call IsSourceReadOnly to determine whether the unit module for the object in the designer is a read-only file.

**Note:** In current version this methods always returns False.

#### 1.15.1.3.1.62 TzCustomFormDesigner.SelectedComponentsCount Method

Returns number of selected components excluding Root (see page 192) and non-component objects.

```
function SelectedComponentsCount: integer;
```

#### 1.15.1.3.1.63 TzCustomFormDesigner.KeyDown Method

Respond to key press events.

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

##### Description

It works similar to TWinControl.KeyDown.

First KeyDown fires OnKeyDown event .

Then processes keystrokes

- VK\_F4 with ssCtrl in Shift switches Active property to False
- VK\_DELETE - deletes selected controls
- VK\_INSERT - with ssCtrl do copying from buffer, with ssShift - pasting
- VK\_DOWN, VK\_UP, VK\_LEFT, VK\_RIGHT - do moving selected controls
- VK\_ESCAPE - moves selection from component to its parent
- VK\_TAB - moves selection to another component in its TabOrder sequences

Finally calls DsnManager.KeyDown procedure.

See TWinControl.KeyDown for details.

#### 1.15.1.3.1.64 TzCustomFormDesigner.KeyPress Method

Respond to keyboard input.

```
procedure KeyPress(var Key: Char); override;
```

##### Description

First KeyPress fires OnKeyPress event .

Finally calls DsnManager.KeyPress procedure.

#### 1.15.1.3.1.65 TzCustomFormDesigner.KeyUp Method

Respond to released key.

```
procedure KeyUp(var Key: Word; Shift: TShiftState); override;
```

##### Description

Invokes OnKeyUp event.

#### 1.15.1.3.1.66 TzCustomFormDesigner.MethodExists Method

Indicates whether an event handler with a specified name already exists.

```
function MethodExists(const Name: string): Boolean;
```

##### Description

Call MethodExists to determine whether an event handler with a given name has already been created.

It checks only existed in executed code methods, i.e. that methods which were created before compilation of application.

#### 1.15.1.3.1.67 TzCustomFormDesigner.Modified Method

Notifies property and component editors when a change is made to a component.

```
procedure Modified;
```

##### Description

When any change is made to a component the property and component editors call this method, allowing the designer to respond to the change.

### 1.15.1.3.1.68 TzCustomFormDesigner.ShowTabOrder Method

Shows tab order icons over children controls of the selected control. Click on the children controls changes their tab order. To exit "Show Tab Icons" mode click on any not child control or press ESCAPE key.

```
procedure ShowTabOrder;
```

### 1.15.1.3.1.69 TzCustomFormDesigner.MouseDown Method

Generates an OnMouseDown event.

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
override;
```

#### Description

It works similar to TControl.MouseDown

See TControl.MouseDown for details.

### 1.15.1.3.1.70 TzCustomFormDesigner.MouseMove Method

Generates an OnMouseMove event.

```
procedure MouseMove(Shift: TShiftState; X: Integer; Y: Integer); override;
```

#### Description

It works similar to TControl.MouseMove

See TControl.MouseMove for details.

### 1.15.1.3.1.71 TzCustomFormDesigner.MouseUp Method

Generates an OnMouseUp event.

```
procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
override;
```

#### Description

It works similar to TControl.MouseUp

See TControl.MouseUp for details.

### 1.15.1.3.1.72 TzCustomFormDesigner.Navigate Method

Performs navigation between control using keyboard

```
procedure Navigate(Key: Word);
```

#### Description

### 1.15.1.3.1.73 TzCustomFormDesigner.UpdateAction Method

Updates an action component to reflect the current state of the component.

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

#### Description

When the application is idle, VCL makes a series of calls to update the properties (such as whether it is enabled, checked,

and so on) of every action that is linked to a visible control or menu item. First, VCL generates an OnUpdate event of the action list that contains the action. If the action list does not handle the OnUpdate event, then the action is routed to the Application object's UpdateAction method, which invokes the OnActionUpdate event handler. If the OnActionUpdate event handler does not update the action, then it is routed to the action's OnUpdate event handler. If that does not update the action, the active control's UpdateAction method is called.

The Action parameter specifies the action component that should be updated. UpdateAction returns true if the action component now reflects the state of the component, and false if it did not know how to update the action. If UpdateAction returns false for the active component, VCL calls the active form's

s UpdateAction method.

Do not call UpdateAction. It is called automatically when the application is idle. As implemented in TComponent, UpdateAction allows the action to update itself with the component as a target. Descendants can override this method to perform updates that reflect class-specific properties or states.

#### 1.15.1.3.1.74 TzCustomFormDesigner.NoSelection Method

Deselects all components in the form designer.

```
procedure NoSelection;
```

##### Description

Use NoSelection to programmatically deselect all components in the designer.

**Note:** To set the selection to a specific component or set of components, use the SelectComponent (see page 185) or SetSelections (see page 186) method.

#### 1.15.1.3.1.75 TzCustomFormDesigner.Undo Method

Backs out last change in the undo buffer.

```
procedure Undo;
```

##### Description

Call Undo to cancel last change made to the Target (see page 193). To determine whether there are any changes in the undo buffer, check the CanUndo (see page 171) property.

#### 1.15.1.3.1.76 TzCustomFormDesigner.Notification Method

Allows the designer to respond when a notification is sent to the form.

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

##### Description

It is overridden procedure from IDesignerNotify.Notification

First it calls inherited, then if AComponent is FPopupMenu it frees FPopupMenu.

#### 1.15.1.3.1.77 TzCustomFormDesigner.NotifySelChanged Method

Notifies active designer about changing selection list

```
procedure NotifySelChanged;
```

**Description****1.15.1.3.1.78 TzCustomFormDesigner.PaintControl Method**

Called for each WM\_PAINT message to perform specific painting over control.

```
function PaintControl(Sender: TControl; var Message: TMessage): Boolean; virtual;
```

**Description****1.15.1.3.1.79 TzCustomFormDesigner.PaintGrid Method**

Paints the alignment grid on the form's canvas.

```
procedure PaintGrid;
```

**Description**

Forms that are associated with a designer call the PaintGrid method in response to Windows paint messages. If there is no designer associated with the form, the form's Paint method is called instead.

**1.15.1.3.1.80 TzCustomFormDesigner.PasteSelection Method**

Pastes the contents of the clipboard into the selected component or components.

```
procedure PasteSelection;
```

**Description**

Call PasteSelection to paste the contents of the clipboard to the components returned by GetSelections ([see page 178](#)).

**1.15.1.3.1.81 TzCustomFormDesigner.ReadComp Method**

Callback procedure for TReader.ReadComponents

```
procedure ReadComp(Component: TComponent);
```

**Description**

ReadComp is a TReadComponentsProc for TReader.ReadComponents procedure.

ReadComponents reads components by calling the ReadComponent method, passing each returned component to the method passed in ReadComp

**1.15.1.3.1.82 TzCustomFormDesigner.RenameMethod Method**

Renames an existing event handler.

```
procedure RenameMethod(const CurName: string; const NewName: string);
```

**Description**

Call RenameMethod to provide an event handler with a new name. The CurName parameter specifies the current name of the event handler, and the NewName parameter specifies the value that the name should be changed to.

This methods calls OnRenameMethod ([see page 200](#)) event to update method name in script code.

**Note:** Before calling RenameMethod, check whether a method with the new name already exists by calling MethodExists ([see page 181](#)).

### 1.15.1.3.1.83 TzCustomFormDesigner.Scale Method

Scale all controls using defined ratio

```
procedure Scale(Ratio: integer);
```

#### Description

Works only if Ratio is between 40 and 400

### 1.15.1.3.1.84 TzCustomFormDesigner.SelectAll Method

Selects all components.

```
procedure SelectAll;
```

#### Description

Selects all components of the form.

### 1.15.1.3.1.85 TzCustomFormDesigner.SelectComponent Method

Replaces the current set of selected components by a single specified object.

```
procedure SelectComponent(Instance: TPersistent); overload;
```

#### Description

Call SelectComponent to select a single persistent object in the designer. Any previous set of selected components is deselected. To select a set of objects, use the SetSelections (see page 186) method instead.

### 1.15.1.3.1.86 TzCustomFormDesigner.SelectedComponent Method

Returns selected component if selection consist of single component; Root (see page 192) component, if there is no selection; nil if selection consists of several components.

```
function SelectedComponent: TComponent;
```

### 1.15.1.3.1.87 TzCustomFormDesigner.SelectionChanged Method

This method call after selection has been changed.

```
procedure SelectionChanged; virtual;
```

#### Description

### 1.15.1.3.1.88 TzCustomFormDesigner.SelectObj Method

Selects object passed as Instance parameter.

```
function SelectObj(Instance: TPersistent; Show: Boolean): Boolean;
```

#### Description

### 1.15.1.3.1.89 TzCustomFormDesigner.SelectRect Method

Selects all controls on the Prn control.

```
procedure SelectRect(ARect: TRect; Prn: TWinControl; ToAdd: Boolean);
```

#### Description

Selects all controls on the Prn control with bounds intersected with ARect. If ToAdd is True controls are added to current selection, otherwise current selection is cleared before adding new control from the ARect.

#### 1.15.1.3.1.90 TzCustomFormDesigner.SendToBack Method

Moves a selected component behind all other components on the form. This is called changing the component's z-order.

```
procedure SendToBack;
```

##### Description

#### 1.15.1.3.1.91 TzCustomFormDesigner.SetPasteName Method

Setting new name for pasting component

```
procedure SetPasteName(Reader: TReader; Component: TComponent; var Name: string); virtual;
```

##### Description

#### 1.15.1.3.1.92 TzCustomFormDesigner.SetScriptEvent Method

Assigns script procedure with particular event of the Instance.

```
procedure SetScriptEvent(Instance: TObject; pInfo: PPropInfo; const EventProc: string);
```

##### Description

This method only calls OnSetScriptEvent handler.

#### 1.15.1.3.1.93 TzCustomFormDesigner.SetSelections Method

Changes the currently selected set of components.

```
procedure SetSelections(const List: IDesignerSelections);
```

##### Description

Call SetSelections to programmatically change which objects in the root object are selected. Pass in a list of selections using an object that supports the IDesignerSelections interface.

**Note:** The List parameter of SetSelections can contain persistent objects that are not components.

#### 1.15.1.3.1.94 TzCustomFormDesigner.ShowMethod Method

Activates the code editor with the input cursor in a specified event handler.

```
procedure ShowMethod(const Name: string);
```

##### Description

Call ShowMethod to allow the user to edit (see page 175) the method specified by the Name parameter.

**Note:** You should write OnShowMethod (see page 201) event handler in which code editor will be activated.

#### 1.15.1.3.1.95 TzCustomFormDesigner.ShowPopupMenu Method

Show designer popup menu at the specified screen position.

```
procedure ShowPopupMenu(X: integer; Y: integer);
```

#### 1.15.1.3.1.96 TzCustomFormDesigner.SizeSelected Method

Perform size operation to selected components

```
procedure SizeSelected(WSize: TCompSize; HSize: TCompSize; AWidth: integer; AHeight: integer);
```

**Description**

It is called after TSizeAdjDlg dialog.

**1.15.1.3.1.97 TzCustomFormDesigner.StartDrag Method**

Makes start settings before dragging operation

```
procedure StartDrag(X: integer; Y: integer);
```

**Description****1.15.1.3.1.98 TzCustomFormDesigner.UniqueName Method**

Generates a unique name from a specified base string.

```
function UniqueName(const BaseName: string): string;
```

**Description**

Call UniqueName to automatically generate a unique name for a component. Specify the base string for the name by the BaseName parameter. UniqueName appends a number to BaseName to ensure that there are no name-space conflicts.

**1.15.1.3.1.99 TzCustomFormDesigner.UpdateCompIcons Method**

Updates positions of component icons.

```
procedure UpdateCompIcons;
```

**Description****1.15.1.3.1.100 TzCustomFormDesigner.ValidateMethod Method**

Determines whether method MAddr of object ARoot is valid method, i.e. it may be assigned to the event.

```
function ValidateMethod(TypeInfo: PTypeInfo; ARoot: TObject; MAddr: pointer; MName: string): Boolean;
```

**1.15.1.3.2 TzCustomFormDesigner Properties****1.15.1.3.2.1 TzCustomFormDesigner.AllowComponents Property**

Specifies whether nonvisual components will be displayed in design mode

```
property AllowComponents: Boolean;
```

**Description**

Nonvisual components (not derived from TControl) are normally invisible in run-time.

AllowComponents specifies whether those components will be displayed in design mode.

**1.15.1.3.2.2 TzCustomFormDesigner.AutoAlign Property**

Specifies using of align rulers.

```
property AutoAlign: Boolean;
```



**Description**

Align rulers are auxiliary lines to nearest controls. They allows easily aligning moved/resized control to other controls.

**1.15.1.3.2.3 TzCustomFormDesigner.BDSStyle Property**

Specifies using of BDS style design environment.

```
property BDSStyle: Boolean;
```

**Description**

When BDSStyle is True moving and resizing of controls are performed as in BDS, i.e. control is visible during operation, otherwise old designer operations are used - when only frames are drawn during moving or resizing.

**1.15.1.3.2.4 TzCustomFormDesigner.CaptionFont Property**

Controls the text attributes of non-visual components captions.

```
property CaptionFont: TFont;
```

**Description**

This property controls the text attributes of non-visual components captions when ShowCaptions (see page 193) is True.

**1.15.1.3.2.5 TzCustomFormDesigner.CloseDisactive Property**

Specifies if TzCustomFormDesigner (see page 161) automatically deactivates when Target (see page 193) Form (see page 189) is to be closed.

```
property CloseDisactive: Boolean;
```

**Description**

Set this property to True to force TzCustomFormDesigner (see page 161) automatically deactivates when Target (see page 193) Form (see page 189) is to be closed.

**1.15.1.3.2.6 TzCustomFormDesigner.ContainerWindow Property**

Determines generic container for any type of Target (see page 193) components

```
property ContainerWindow: TWinControl;
```

**Description**

ContainerWindow is used internally to represent common generic approach for Target (see page 193) component.

Do not specify ContainerWindow directly, use Target (see page 193) property instead.

**1.15.1.3.2.7 TzCustomFormDesigner.DesignSurface Property**

Specifies design surface.

```
property DesignSurface: TDesignSurface;
```

**Description**

When design surface, control of TDesignSurface (see page 237) class, is assigned form is placed on it. Form (see page 189) does not activated, when user clicks on the form design surface gets focus.

Design surface is usual control which can not be placed on any container - form, tab sheet, panel, etc.

This allows organizing of multiple documents applications where each form is placed on separate tab sheet.

Also design surface allows hiding of form caption and border by setting TDesignSurface.HideFormBorders (see page 240) property.

Using design surface you may implement BDS like design environment.

#### 1.15.1.3.2.8 TzCustomFormDesigner.DisplayControlGrid Property

Specifies whether designer has to paint grid over window controls that can accept controls.

**property** DisplayControlGrid: Boolean;

##### Description

Set this property to True to display design grid over window controls that can accept controls, for example, over tab sheet or panel.

#### 1.15.1.3.2.9 TzCustomFormDesigner.DisplayGrid Property

Determines whether dots are drawn on the Target (see page 193) form.

**property** DisplayGrid: Boolean;

##### Description

Set DisplayGrid to True for dots representing as grid on the Target (see page 193) Form (see page 189).

Dots will be shown only if Target (see page 193) is a TCustomForm descendant.

#### 1.15.1.3.2.10 TzCustomFormDesigner.FlatIcons Property

Determines whether the non-visual component icons has a 3D border or not.

**property** FlatIcons: Boolean;

##### Description

Set FlatIcons to True to remove the 3D border around the non-visual component icons.

#### 1.15.1.3.2.11 TzCustomFormDesigner.DragParentLimit Property

Specifies whether drag mouse movement should be clipped by parent's client area.

**property** DragParentLimit: Boolean;

#### 1.15.1.3.2.12 TzCustomFormDesigner.Form Property

Provides access to designed form as TCustomForm type.

**property** Form: TCustomForm;

##### Description

Form is used internally to represent specific approach for Target (see page 193) component.

Do not specify Form directly, use Target (see page 193) property instead.

#### 1.15.1.3.2.13 TzCustomFormDesigner.Events Property

Storage of assigned events.

**property** Events: TStrings;

##### Description

Designer stores assigned events (associations between event and script procedure) to this string list. To enable events processing by designer set property StoreEvents (see page 191) to True.

Use Events.Names[Index] to access property path in view "ComponentName.EventName". For root events component name is omitted. Use Events.ValueFromIndex[Index] to get script procedure name.

#### 1.15.1.3.2.14 TzCustomFormDesigner.GridStepX Property

Specifies grid step, in pixels, along X-axis

```
property GridStepX: integer;
```

##### Description

Use this property to read or change grid step along X-axis.

#### 1.15.1.3.2.15 TzCustomFormDesigner.GridStepY Property

Specifies grid step, in pixels, along Y-axis

```
property GridStepY: integer;
```

##### Description

Use this property to read or change grid step along Y-axis.

#### 1.15.1.3.2.16 TzCustomFormDesigner.LockControls Property

Specifies if user can directly change size and position of controls by mouse.

```
property LockControls: Boolean;
```

##### Description

Set this property to False to allow user changing controls position and size only through Object Inspector (TObjectInspector).

#### 1.15.1.3.2.17 TzCustomFormDesigner.LockPublished Property

Specifies if editing operation are forbidden by default

```
property LockPublished: Boolean;
```

##### Description

Set this property to True to programmatically control what kind of editing will be allowed.

User can adjust reaction on editing request in event handlers.

It affects on events

- OnCanDelete (see page 195)
- OnCanSelect (see page 196)
- OnCanResize (see page 196)
- OnCanMove (see page 196)
- OnCanRename (see page 196)

#### 1.15.1.3.2.18 TzCustomFormDesigner.MultiSelect Property

Determines whether the user can select more than one control at a time.

```
property MultiSelect: Boolean;
```

##### Description

Set MultiSelect to True to allow the user to select multiple controls. If MultiSelect if False, multiple controls cannot be selected at the same time.

#### 1.15.1.3.2.19 TzCustomFormDesigner.Groups Property

Stores groups information.

```
property Groups: TControlGroups;
```

**Description**

Grouping controls allows easier selection and operations on groups of controls.

**1.15.1.3.2.20 TzCustomFormDesigner.GuidelinesStyle Property**

Specifies guidelines options.

```
property GuidelinesStyle: TGuidelinesStyles;
```

**1.15.1.3.2.21 TzCustomFormDesigner.IgnoreReadErrors Property**

Specifies whether read errors should be ignored when loading using LoadFromFile (see page 177) and LoadFromStream (see page 178) methods.

```
property IgnoreReadErrors: Boolean;
```

**1.15.1.3.2.22 TzCustomFormDesigner.ReadOnly Property**

Set Read Only mode.

```
property ReadOnly: Boolean;
```

**Description**

Set ReadOnly to True to disable any changes in designer. In read only mode there is no popup menu, all changes to controls are disabled (moving, resizing, etc.), component editors are disabled.

**1.15.1.3.2.23 TzCustomFormDesigner.StoreEvents Property**

Specifies whether designer should process events storage.

```
property StoreEvents: Boolean;
```

**Description**

When StoreEvents is True events associations are save to Events (see page 189) property. These events are saved in resource (DFM file or stream).

Otherwise you need to use OnSetScriptProc (see page 200) and OnGetScriptProc (see page 199) events to process events associations manually.

**1.15.1.3.2.24 TzCustomFormDesigner.TabOrderIcons Property**

Properties of tab order icons. These controls are shown over children control when designer is in "Show Tab Order" mode.

```
property TabOrderIcons: TTabOrderIcons;
```

**1.15.1.3.2.25 TzCustomFormDesigner.TextEditMode Property**

Sets in-place text editing mode of designer.

```
property TextEditMode: Boolean;
```

**Description**

Set TextEditMode to True to enable in-place editing of control's texts. In this mode labels, button captions, list box items, combo box items and many others text properties may be edited directly on the form. It will give more interactivity to design process.

To activate in-place editor user needs to click on text string on the control or press Enter, when this control is selected.

In-place editors are handled by special design classes derived from

**1.15.1.3.2.26 TzCustomFormDesigner.PopupMenu Property**

Identifies the pop-up menu associated with the Root (see page 192) control of the Designer.

```
property PopupMenu: TPopupMenu;
```

#### Description

It is similar to TControl.PopupMenu property.

If there are no popup menu assigned to this property default popup menu is created. In this case you may use PopupMenuFilter (see page 192) to define possible item groups in default popup menu.

See TControl.PopupMenu for details.

### 1.15.1.3.2.27 TzCustomFormDesigner.UndoLimit Property

Specifies the number of changes that can be undone.

```
property UndoLimit: integer;
```

#### Description

Use UndoLimit to restrict undo (see page 183) records list. If UndoLimit is 0, undo (see page 183) operation is disabled.

Default value is 16.

### 1.15.1.3.2.28 TzCustomFormDesigner.PopupMenuFilter Property

Specifies which categories of items may be used to form default popup menu.

```
property PopupMenuFilter: TLocalMenuFilters;
```

#### Description

### 1.15.1.3.2.29 TzCustomFormDesigner.UndoLoad Property

Indicates that designer is in Undo (see page 183) loading state, i.e. in reading previously saved form resource.

```
property UndoLoad: Boolean;
```

### 1.15.1.3.2.30 TzCustomFormDesigner.Root Property

Root component for TzCustomFormDesigner (see page 161).

```
property Root: TComponent;
```

#### Description

Designer works only with components owned by the root.

Root is used internally to represent specific approach for Target (see page 193) component.

Do not specify Root directly, use Target (see page 193) property instead.

### 1.15.1.3.2.31 TzCustomFormDesigner.RootModified Property

Indicates whether the Root (see page 192) or its components are modified (see page 181).

```
property RootModified: Boolean;
```

#### Description

Check Modified (see page 181) to determine if the designed components are modified (see page 181). Write it to force OnRootModified event.

### 1.15.1.3.2.32 TzCustomFormDesigner.SelCount Property

Indicates number of selected components.

```
property SelCount: integer;
```

#### Description

Read this property to get the number of components selected in Root (see page 192).

### 1.15.1.3.2.33 TzCustomFormDesigner.Selected Property

Indicates whether a particular control is selected.

```
property Selected [Index: integer]: TControl;
```

#### Description

Use Selected to get the indexed control from the designer's array of selected items.

The Index parameter is the item referenced by its position in the array of controls, with the first item having an Index value of 0.

### 1.15.1.3.2.34 TzCustomFormDesigner.SelMarker Property

Selection markers manager.

```
property SelMarker: TzBoundCtrl;
```

#### Description

Use this object to customize shape and colors of selection markers. Selection markers are visible when only one component is selected.

### 1.15.1.3.2.35 TzCustomFormDesigner.ShowCaptions Property

Specifies whether non-visual component icons captions are visible.

```
property ShowCaptions: Boolean;
```

#### Description

When ShowCaption is True non-visual components icons have captions with their names below them.

For data modules captions are always visible

### 1.15.1.3.2.36 TzCustomFormDesigner.SnapToGrid Property

Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines.

```
property SnapToGrid: Boolean;
```

#### Description

### 1.15.1.3.2.37 TzCustomFormDesigner.Target Property

Specifies object that is edited by the designer.

```
property Target: TComponent;
```

#### Description

Setting this property affects to Root (see page 192), ContainerWindow (see page 188) and Form (see page 189) properties.

**Possible Target types are following:**

1. TCustomForm - form designing.

**In this case:**

Root (see page 192) = ContainerWindow (see page 188) = Form (see page 189) = Target.

2. TWinControl, owned by the form. This control and all it's children controls are edited. Also non-visual components owned by the form may be edited too if AllowComponents (see page 187) is True.

**In this case:**

Root (see page 192) = Form (see page 189) = (Target.Owner as TCustomForm);

ContainerWindow (see page 188) = (Target as TWinControl);

3. TWinControl, not owned by the form, for example, TQuickReport.

This control and all owned by it components are edited.

In this case temporary form is created as container in design mode.

Client size of this form is equal to size of the Target.

Changing form size does not change size of the Target.

**In this case:**

Form (see page 189) = temporary internal Form (see page 189);

Root (see page 192) = Target;

ContainerWindow (see page 188) = (Target as TWinControl);

4. TCustomFrame.

Identical to case 3, but changing form size changes size of the frame.

5. TDataModule.

Data module and all it's components are edited.

For editing of data modules temporary Form (see page 189) and temporary container are created, in which designing are performed.

**In this case:**

Root (see page 192) = Target;

Form (see page 189) = temporary internal Form (see page 189);

ContainerWindow (see page 188) = temporary internal Container;

### 1.15.1.3.3 TzCustomFormDesigner Events

#### 1.15.1.3.3.1 TzCustomFormDesigner.OnCanEdit Event

Occurs to determine whether Edit (see page 175) method of component editor can be called.

```
property OnCanEdit: TComponentEvent;
```

**Description**

Write OnCanEdit event handler to disable component editor call when user double clicks component.

**1.15.1.3.3.2 TzCustomFormDesigner.OnDrawControl Event**

Occurs when painting any control on the form. Use this event to draw over control.

```
property OnDrawControl: TDrawControlEvent;
```

**1.15.1.3.3.3 TzCustomFormDesigner.OnExecuteAction Event**

Occurs when ExecuteAction (see page 175) method is called. Use it to handle standard shared actions for currently active designer.

```
property OnExecuteAction: THandleActionEvent;
```

**1.15.1.3.3.4 TzCustomFormDesigner.OnGetComponentLocked Event**

Occurs to determine whether component is locked.

```
property OnGetComponentLocked: TComponentEvent;
```

**Description**

Write OnGetComponentLocked event handler to specify which components are locked, i.e can be edited in designer.

**1.15.1.3.3.5 TzCustomFormDesigner.OnCanDelete Event**

Occurs before deleting selected component.

```
property OnCanDelete: TComponentEvent;
```

**Description**

Write an OnCanDelete event handler to provide custom action. You may forbid deleting for example.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being deleted

The Accept parameter determines whether a component is allowed to delete.

**1.15.1.3.3.6 TzCustomFormDesigner.OnCanInsert Event**

Occurs before inserting new component.

```
property OnCanInsert: TComponentEvent;
```

**Description**

Write an OnCanInsert event handler to provide custom action. You may forbid inserting this new component.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being inserted

The Accept parameter determines whether a component is allowed to insert.

**1.15.1.3.3.7 TzCustomFormDesigner.OnGetObjectName Event**

Occurs at the end of GetObjectName (see page 177) method to adjust resulting name.

```
property OnGetObjectName: TGetObjNameEvent;
```



**Description**

This event may be helpful if visible control is only proxy of real object.

**1.15.1.3.3.8 TzCustomFormDesigner.OnCanMove Event**

Occurs before moving selected component.

**property** OnCanMove: TComponentEvent;

**Description**

Write an OnCanMove event handler to provide custom action. You may forbid moving this component.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being moved

The Accept parameter determines whether a component is allowed to move.

**1.15.1.3.3.9 TzCustomFormDesigner.OnCanRename Event**

Occurs before renaming component.

**property** OnCanRename: TRenameEvent;

**Description**

Write an OnCanRename event handler to provide custom action. You may forbid renaming this component or set another name.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being renamed

The NewName parameter is the suggested name

The Accept parameter determines whether a component is allowed to rename.

**1.15.1.3.3.10 TzCustomFormDesigner.OnCanResize Event**

Occurs before resizing selected component.

**property** OnCanResize: TComponentEvent;

**Description**

Write an OnCanResize event handler to provide custom action. You may forbid resizing this component.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being resized

The Accept parameter determines whether a component is allowed to resize.

**1.15.1.3.3.11 TzCustomFormDesigner.OnPopUndo Event**

Occurs when restoring Target (see page 193) from undo (see page 183) buffer.

**property** OnPopUndo: TUndoRecEvent;

**1.15.1.3.3.12 TzCustomFormDesigner.OnCanSelect Event**

Occurs before selecting component.

**property** OnCanSelect: TComponentEvent;

#### Description

Write an OnCanSelect event handler to provide custom action. You may forbid selecting this component.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being selected

The Accept parameter determines whether a component is allowed to select.

### 1.15.1.3.3.13 TzCustomFormDesigner.OnPushUndo Event

Occurs when saving Target (see page 193) to undo (see page 183) buffer.

**property** OnPushUndo: TUndoRecEvent;

### 1.15.1.3.3.14 TzCustomFormDesigner.OnCreateComponent Event

Occurs before new component creation.

**property** OnCreateComponent: TCreateComponentEvent;

#### Description

Write this handler to customize component creation.

AClass - class of inserted component;

AOwner - owner of the component (root)

Instance - reference to component to be created.

**Note:** if you create (see page 174) event handler you should create (see page 174) component in it. Default component creation will be skipped. This allows to restrict component creation.

### 1.15.1.3.3.15 TzCustomFormDesigner.OnCreateFrame Event

Occurs when frame is to be inserted on the form.

**property** OnCreateFrame: TCreateFrameEvent;

#### Description

By default PackageMng.CreateFrame method is called. This method shows select frame dialog and creates an instance of frame class. Using this event you may change default behavior.

### 1.15.1.3.3.16 TzCustomFormDesigner.OnSetNewName Event

Occurs when assigning name to newly inserted component (created or pasted).

**property** OnSetNewName: TSetNameEvent;

### 1.15.1.3.3.17 TzCustomFormDesigner.OnCreatelcon Event

Occurs before creating icon for non-visual component.

**property** OnCreateIcon: TCreateIconEvent;

#### Description

Write an OnCreatelcon event handler to provide custom action. You may forbid component icon creation, so that component will not be visible on the designed Target (see page 193), but will be accessible in object inspector.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component icon is being created for

The AllowCreate parameter determines whether an icon will be created.

#### 1.15.1.3.3.18 TzCustomFormDesigner.OnCreateMethod Event

Occurs when new method name is input in object inspector.

**property** OnCreateMethod: TCreateMethodEvent;

##### Description

Write this event handler to specify method reference when there is no registered method with the specified Name.

Use TypeData to validate possible assignment, i.e. do not assign to Method reference to procedure with another type. This may cause system halt.

Note: This event is intended for only in-code implemented methods. If you are working with script procedures use OnSetScriptProc (see page 200) event.

#### 1.15.1.3.3.19 TzCustomFormDesigner.OnFormClosed Event

Occurs immediately after hiding the Target (see page 193) form .

**property** OnFormClosed: TNotifyEvent;

##### Description

Write an OnFormClosed event handler to provide custom action when designed form is closed.

The Sender parameter is the object whose event handler is called.

#### 1.15.1.3.3.20 TzCustomFormDesigner.OnUpdateAction Event

Occurs when UpdateAction (see page 182) method is called. Use it to handle standard shared actions for currently active designer.

**property** OnUpdateAction: THandleActionEvent;

#### 1.15.1.3.3.21 TzCustomFormDesigner.OnGetComponentHint Event

Occurs when the application is about to display the hint window for the particular component.

**property** OnGetComponentHint: TGetComponentHintEvent;

##### Description

Write an OnGetComponentHint event handler to provide custom action. You may change hint or even forbid to display it.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being selected

The Hint parameter is the hint string that user can change

The ShowHint parameter determines whether a hint will be displayed.

#### 1.15.1.3.3.22 TzCustomFormDesigner.OnGetMethodNames Event

Occurs when method property editor requests designer for possible method names.

**property** OnGetMethodNames: TGetMethodNamesEvent;

**Description**

Write this handler to specify possible procedure names that can be assigned to the selected in object inspector event.

Use TypeData to filter possible procedures, for example, by procedure parameters. Call Proc for each possible procedure.

This event is intended for integration with external script engines.

**See Also**

OnGetScriptProc (see page 199), OnSetScriptProc (see page 200), OnShowMethod (see page 201), OnRenameMethod (see page 200)

**Example**

This example shows using of the event. EControl Syntax Editor SDK is used as source code analyzer

```
procedure TForm4.zFormDesigner1GetMethodNames(Sender: TObject;
  TypeData: PTypeData; Proc: TGetStrProc);
var i: integer;
    R: TTagBlockCondition;
begin
  with CodeEditor.SyntObj do
  begin
    // Looking for all text ranges with rule "function"
    R := TTagBlockCondition(Owner.BlockRules.ItemByName('function'));
    if R <> nil then
      for i := 0 to RangeCount - 1 do
        if (Ranges[i].Rule = R) then
          Proc(TagStr[Ranges[i].StartIdx + 1]); // Adds function to procedures list
      end;
    end;
  end;
end;
```

**1.15.1.3.3.23 TzCustomFormDesigner.OnGetScriptProc Event**

Occurs when method property editor ask for script procedure name associated with a given property.

```
property OnGetScriptProc: TGetScriptProcEvent;
```

**Description**

Write this event handler to return procedure name associated with property *pInfo^.Name* of object *Instance*.

**See Also**

OnGetMethodNames (see page 198), OnSetScriptProc (see page 200), OnShowMethod (see page 201), OnRenameMethod (see page 200)

**Example**

In this sample associations of script procedures and properties are stores in TStrings object (property Items in list box EventsList) with items - <ObjectName>.<PropertyName>=<ScriptProcedure>

```
procedure TForm4.zFormDesigner1GetScriptProc(Sender, Instance: TObject;
  pInfo: PPropInfo; var ProcName: String);
begin
  ProcName := '';
  if Instance is TComponent then
    if zFormDesigner1.Root = Instance then
      ProcName := EventsList.Items.Values[pInfo^.Name]
    else
      ProcName := EventsList.Items.Values[(Instance as TComponent).Name + '.' +
        pInfo^.Name];
  end;
end;
```

**1.15.1.3.3.24 TzCustomFormDesigner.OnNotification Event**

Occurs when components are added or removed to/from Root (see page 192) object at design mode.

```
property OnNotification: TNotificationEvent;
```

**Description**

Write an OnNotification event handler to provide custom action when components are added or removed to/from Root (see page 192) object at design mode.

The Sender parameter is the object whose event handler is called.

The AnObject parameter is the TPersistent object that is being operated

The Operation parameter specifies kind of operation applied to object

- opInsert - object inserted
- opRemove - object removed.

**1.15.1.3.3.25 TzCustomFormDesigner.OnRenameMethod Event**

Occurs when name of method is changed in object inspector.

**property** OnRenameMethod: TRenameMethodEvent;

**Description**

Write this event handler to change script procedure name in script code. You should find script procedure CurName and change its name to NewName.

**See Also**

OnGetMethodNames (see page 198), OnSetScriptProc (see page 200), OnShowMethod (see page 201), OnGetScriptProc (see page 199)

**1.15.1.3.3.26 TzCustomFormDesigner.OnSetScriptProc Event**

Occurs when method property editor assigns script procedure to the event.

**property** OnSetScriptProc: TSetScriptProcEvent;

**Description**

Write this event handler to save association between event property end script procedure.

**See Also**

OnGetMethodNames (see page 198), OnRenameMethod (see page 200), OnShowMethod (see page 201), OnGetScriptProc (see page 199)

**Example**

In this sample associations of script procedures and properties are stores in TStrings object (property Items in list box EventsList) with items - <ObjectName>.<PropertyName>=<ScriptProcedure>

```
procedure TForm4.zFormDesigner1SetScriptProc(Sender, Instance: TObject;
  pInfo: PPropInfo; const EventProc: String);
var idx: integer;
    pn: string;
begin
  if Instance is TComponent then
    begin
      // event name for root object is without object name
      if zFormDesigner1.Root = Instance then
        pn := pInfo^.Name
      else
        pn := (Instance as TComponent).Name + '.' + pInfo^.Name;
      // Delete previous association
      idx := EventsList.Items.IndexOfName(pn);
      if idx <> -1 then
        EventsList.Items.Delete(idx);
      // Add new association
      if EventProc <> '' then
```

```

begin
  // Saving associating
  EventsList.Items.Add(pn + '=' + EventProc);
  // Creating event handler text body
  CreateMethod(EventProc, Instance, pInfo);
end;
end;
end;

```

### 1.15.1.3.3.27 TzCustomFormDesigner.OnShowMethod Event

Occurs when user double clicks on the procedure in the object inspector.

**property** OnShowMethod: TShowMethodEvent;

#### Description

Write this handler to highlight script procedure MethodName in script code text.

### 1.15.1.3.3.28 TzCustomFormDesigner.OnValidateMethod Event

Occurs to validate method.

**property** OnValidateMethod: TValidateMethodEvent;

#### Description

Write an OnValidateMethod event handler to provide custom validation of the particular method.

The Sender parameter is the object whose event handler is called.

The TypeData parameter is the type of method

The ARoot parameter is the owner of this method

The MethAddr parameter is the address of this method

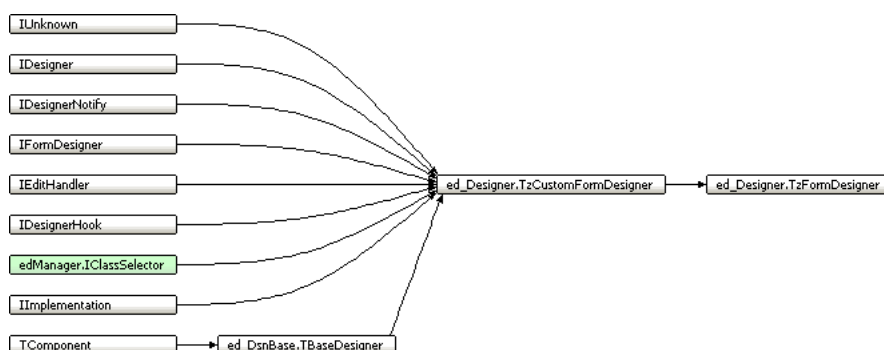
The MethodName parameter is the name of method

The Accept parameter determines whether this method is allowed to add to list of methods.

## 1.15.1.4 TzFormDesigner Class

TzFormDesigner is main library component.

#### Class Hierarchy



```
TzFormDesigner = class(TzCustomFormDesigner);
```

#### File

ed\_Designer

## Description

TzFormDesigner provides full-sized functionality to manipulate with controls at run-time as if it was to be in design-time IDE.























TzFormDesigner implements all the required design-time interfaces for this purpose and propose a set of properties, methods and events to create (see page 174) design-like environment at runtime.

With TzFormDesigner user may design forms, data-modules, report sheets and so on.


It uses all the standard Borland property and component editors.

## Members












### TBaseDesigner Methods

TBaseDesigner Methods	Description
 CanProcessNCMessages (see page 146)	Called to check processing of non client mouse messages.
 Client2Screen (see page 147)	Corrected version of ClientToScreen function to support RTL.
 ClientOrg (see page 147)	Corrected version of ClientOrigin function to support RTL.
 Create (see page 147)	Creates and initializes a TBaseDesigner instance.
 DesignState (see page 147)	Indicates whether component in design or loading state. At design time (in Delphi IDE) and during loading no activation occurs.
 Destroy (see page 147)	Destroys an instance of TBaseDesigner.
 DoObjectHint (see page 147)	Called to show hint for current design state.
 DragDrop (see page 147)	OnDragDrop (see page 152) event dispatcher.
 DragOver (see page 148)	OnDragOver (see page 152) event dispatcher.
 IsRTL (see page 148)	Returns true when control is right-to-left.
 KeyDown (see page 148)	Respond to key press events.
 KeyPress (see page 149)	Respond to keyboard input.
 KeyUp (see page 149)	Respond to released key.
 Loaded (see page 149)	Initializes the component after the form file has been read into memory.
 MouseDown (see page 150)	OnMouseDown (see page 154) event dispatcher.
 MouseMove (see page 150)	OnMouseMove (see page 154) event dispatcher.
 MouseUp (see page 150)	OnMouseUp (see page 154) event dispatcher.
 ProcessMessage (see page 151)	Translates messages of all managed by designer controls.
 ResetHint (see page 151)	Resets hint for another component
 Screen2Client (see page 151)	Corrected version of ScreenToClient function to support RTL.
 SetActive (see page 151)	Set method for Active (see page 151) property.
 ShowHint (see page 151)	Indicates whether hints should be shown for controls in the Designer.

### IClassSelector Interface

IClassSelector Interface	Description
 ClsChanged (see page 511)	Called when selected in component palette component class was changed.
 ClsPalChanged (see page 512)	Called when component palette was changed.

### TzCustomFormDesigner Class

TzCustomFormDesigner Class	Description
 AddCompEditorMenu (see page 170)	Adds menu items to the popup Menu associated with the component editor of selected in designer component. To remove previously added component editor's menu items use ClearCompEditorMenu (see page 171) method.
 CanRedo (see page 170)	Indicates whether the designer contains undone changes that can be repeated.
 CanUndo (see page 171)	Indicates whether the designer contains changes that can be backed out.
 CheckAction (see page 171)	Determines whether design operation specified by the Action parameter can be executed.
 ClearCompEditorMenu (see page 171)	Removes component editor's menu items added by the AddCompEditorMenu (see page 170) method.
 ClearUndo (see page 171)	Clears the undo (see page 183) buffer so that no changes to the Target (see page 193) can be backed out.
 CloseTextEditor (see page 171)	Closes in-place editor.
 DragDrop (see page 171)	OnDragDrop event dispatcher.
 DragOver (see page 171)	OnDragOver event dispatcher.
 EditAction (see page 172)	Executes designer operation specified by the Action parameter.
 AlignSelected (see page 172)	Performs alignment operation on selected components.

◆ AlignToGrid (see page 172)	Aligns selected components to the closest grid point.
◆ BringToFront (see page 172)	Moves a selected component in front of all other components on the form. This is called changing the component's z-order.
◆ BuildLocalMenu (see page 172)	Creates default popup menu.
◆ CancelDrag (see page 173)	Cancels current drag&drop design operation.
◆ CanDelete (see page 173)	Check if component can be deleted.
◆ GetEditState (see page 173)	Returns set of possible designer operations.
◆ CanInsert (see page 173)	Check if component can be inserted.
◆ CanMove (see page 173)	Check if component can be moved.
◆ CanPaste (see page 173)	Checks clipboard for components saved to it.
◆ CanRename (see page 173)	Check if component can be renamed.
◆ CanResize (see page 174)	Check if component can be resized.
◆ CanSelect (see page 174)	Check if component can be selected.
◆ ClearSelection (see page 174)	Resets selection.
◆ CopySelection (see page 174)	
◆ Create (see page 174)	Creates and initializes a TzCustomFormDesigner instance.
◆ CutSelection (see page 174)	Cuts selected components to clipboard.
◆ DeleteSelection (see page 175)	Deletes the selected component or components
◆ Destroy (see page 175)	Destroys an instance of TzCustomFormDesigner.
◆ DoObjectHint (see page 175)	Shows hint current design state.
◆ ExecuteAction (see page 175)	Invokes an action with the component as its target.
◆ DragDraw (see page 175)	Provides drag-and-draw graphic operation.
◆ Edit (see page 175)	Displays the component editor for the specified component.
◆ EndDrag (see page 176)	Provides drag-and-draw graphic operation.
◆ FlipChildren (see page 176)	Allows to reverse the layout of components in the current form to a right-to-left mirror image.
◆ GetCompObj (see page 176)	Returns component
◆ GetComponent (see page 176)	Returns the component with the name passed as a parameter.
◆ GetComponentName (see page 176)	Returns the name of the component passed as its parameter.
◆ GetComponentNames (see page 177)	Executes a callback for every component that can be assigned a property of a specified type.
◆ GetControlAt (see page 177)	Looks for control at specified point on the window prn.
◆ GetMethodName (see page 177)	Returns the name of a specified event handler.
◆ GetNewName (see page 177)	Returns new generated name for particular class of the component.
◆ GetObjectName (see page 177)	Returns object name.
◆ LoadFromFile (see page 177)	Load Root (see page 192) component from file. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.
◆ LoadFromStream (see page 178)	Load Root (see page 192) component from stream. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.
◆ GetRoot (see page 178)	Returns the current entity being edited by the form designer.
◆ GetRootClassName (see page 178)	Returns the class name for the root component.
◆ GetScriptEvent (see page 178)	Returns script procedure name assigned to the event.
◆ GetScrollRanges (see page 178)	Returns the size of the logical designer window.
◆ GetSelections (see page 178)	Fills a list with all selected components on the current root object.
◆ Redo (see page 179)	Call Redo to repeat last undone operation.
◆ GetShiftState (see page 179)	Returns the current state of the Shift, Alt, and Ctrl keys.
◆ Intf_Notification (see page 179)	Allows the designer to respond when a notification (see page 183) is sent to the form.
◆ IsComponentHidden (see page 179)	Indicates whether a component does not appear directly in the form designer.
◆ IsLocked (see page 179)	Specifies whether component is locked and can not be edited in designer.
◆ SaveToFile (see page 179)	Saves Root (see page 192) component to file FileName with events information. AsText specifies format of the file: text or binary.
◆ IsDesignMsg (see page 180)	Determines when the designer should handle a Windows message.
◆ SaveToStream (see page 180)	Save Root (see page 192) component to stream with events information. AsText specifies storage format: text or binary.
◆ IsProtected (see page 180)	Indicates whether component is protected and can not be changed by designer.
◆ IsRootSelected (see page 180)	Returns True if entire Root (see page 192) is selected



IsSourceReadOnly (see page 180)	Indicates whether the source file for the component being designed is read-only.
SelectedComponentsCount (see page 180)	Returns number of selected components excluding Root (see page 192) and non-component objects.
KeyDown (see page 180)	Respond to key press events.
KeyPress (see page 181)	Respond to keyboard input.
KeyUp (see page 181)	Respond to released key.
MethodExists (see page 181)	Indicates whether an event handler with a specified name already exists.
Modified (see page 181)	Notifies property and component editors when a change is made to a component.
ShowTabOrder (see page 182)	Shows tab order icons over children controls of the selected control. Click on the children controls changes their tab order. To exit "Show Tab Icons" mode click on any not child control or press ESCAPE key.
MouseDown (see page 182)	Generates an OnMouseDown event.
MouseMove (see page 182)	Generates an OnMouseMove event.
MouseUp (see page 182)	Generates an OnMouseUp event.
Navigate (see page 182)	Performs navigation between control using keyboard
UpdateAction (see page 182)	Updates an action component to reflect the current state of the component.
NoSelection (see page 183)	Deselects all components in the form designer.
Undo (see page 183)	Backs out last change in the undo buffer.
Notification (see page 183)	Allows the designer to respond when a notification is sent to the form.
NotifySelChanged (see page 183)	Notifies active designer about changing selection list
PaintControl (see page 184)	Called for each WM_PAINT message to perform specific painting over control.
PaintGrid (see page 184)	Paints the alignment grid on the form's canvas.
PasteSelection (see page 184)	Pastes the contents of the clipboard into the selected component or components.
ReadComp (see page 184)	Callback procedure for TReader.ReadComponents
RenameMethod (see page 184)	Renames an existing event handler.
Scale (see page 185)	Scale all controls using defined ratio
SelectAll (see page 185)	Selects all components.
SelectComponent (see page 185)	Replaces the current set of selected components by a single specified object.
SelectedComponent (see page 185)	Returns selected component if selection consist of single component; Root (see page 192) component, if there is no selection; nil if selection consists of several components.
SelectionChanged (see page 185)	This method call after selection has been changed.
SelectObj (see page 185)	Selects object passed as Instance parameter.
SelectRect (see page 185)	Selects all controls on the Prn control.
SendToBack (see page 186)	Moves a selected component behind all other components on the form. This is called changing the component's z-order.
SetPasteName (see page 186)	Setting new name for pasting component
SetScriptEvent (see page 186)	Assigns script procedure with particular event of the Instance.
SetSelections (see page 186)	Changes the currently selected set of components.
ShowMethod (see page 186)	Activates the code editor with the input cursor in a specified event handler.
ShowPopupMenu (see page 186)	Show designer popup menu at the specified screen position.
SizeSelected (see page 186)	Perform size operation to selected components
StartDrag (see page 187)	Makes start settings before dragging operation
UniqueName (see page 187)	Generates a unique name from a specified base string.
UpdateComplcons (see page 187)	Updates positions of component icons.
ValidateMethod (see page 187)	Determines whether method MAddr of object ARoot is valid method, i.e. it may be assigned to the event.

### TBaseDesigner Properties












TBaseDesigner Properties	Description
Active (see page 151)	Switches target component between design and run-time modes
HintObject (see page 152)	Specifies object for which hint was activated.
ShowHints (see page 152)	Specifies showing of design hints.




















































### TzCustomFormDesigner Class







TzCustomFormDesigner Class	Description
AllowComponents (see page 187)	Specifies whether nonvisual components will be displayed in design mode
AutoAlign (see page 187)	Specifies using of align rulers.
BDSStyle (see page 188)	Specifies using of BDS style design environment.

 <b>CaptionFont</b> ( <a href="#">see page 188</a> )	Controls the text attributes of non-visual components captions.
 <b>CloseDisactive</b> ( <a href="#">see page 188</a> )	Specifies if TzCustomFormDesigner ( <a href="#">see page 161</a> ) automatically deactivates when Target ( <a href="#">see page 193</a> ) Form ( <a href="#">see page 189</a> ) is to be closed.
 <b>ContainerWindow</b> ( <a href="#">see page 188</a> )	Determines generic container for any type of Target ( <a href="#">see page 193</a> ) components
 <b>DesignSurface</b> ( <a href="#">see page 188</a> )	Specifies design surface.
 <b>DisplayControlGrid</b> ( <a href="#">see page 189</a> )	Specifies whether designer has to paint grid over window controls that can accept controls.
 <b>DisplayGrid</b> ( <a href="#">see page 189</a> )	Determines whether dots are drawn on the Target ( <a href="#">see page 193</a> ) form.
 <b>FlatIcons</b> ( <a href="#">see page 189</a> )	Determines whether the non-visual component icons has a 3D border or not.
 <b>DragParentLimit</b> ( <a href="#">see page 189</a> )	Specifies whether drag mouse movement should be clipped by parent's client area.
 <b>Form</b> ( <a href="#">see page 189</a> )	Provides access to designed form as TCustomForm type.
 <b>Events</b> ( <a href="#">see page 189</a> )	Storage of assigned events.
 <b>GridStepX</b> ( <a href="#">see page 190</a> )	Specifies grid step, in pixels, along X-axis
 <b>GridStepY</b> ( <a href="#">see page 190</a> )	Specifies grid step, in pixels, along Y-axis
 <b>LockControls</b> ( <a href="#">see page 190</a> )	Specifies if user can directly change size and position of controls by mouse.
 <b>LockPublished</b> ( <a href="#">see page 190</a> )	Specifies if editing operation are forbidden by default
 <b>MultiSelect</b> ( <a href="#">see page 190</a> )	Determines whether the user can select more than one control at a time.
 <b>Groups</b> ( <a href="#">see page 190</a> )	Stores groups information.
 <b>GuidelinesStyle</b> ( <a href="#">see page 191</a> )	Specifies guidelines options.
 <b>IgnoreReadErrors</b> ( <a href="#">see page 191</a> )	Specifies whether read errors should be ignored when loading using LoadFromFile ( <a href="#">see page 177</a> ) and LoadFromStream ( <a href="#">see page 178</a> ) methods.
 <b>ReadOnly</b> ( <a href="#">see page 191</a> )	Set Read Only mode.
 <b>StoreEvents</b> ( <a href="#">see page 191</a> )	Specifies whether designer should process events storage.
 <b>TabOrderIcons</b> ( <a href="#">see page 191</a> )	Properties of tab order icons. These controls are shown over children control when designer is in "Show Tab Order" mode.
 <b>TextEditMode</b> ( <a href="#">see page 191</a> )	Sets in-place text editing mode of designer.
 <b>PopupMenu</b> ( <a href="#">see page 191</a> )	Identifies the pop-up menu associated with the Root ( <a href="#">see page 192</a> ) control of the Designer.
 <b>UndoLimit</b> ( <a href="#">see page 192</a> )	Specifies the number of changes that can be undone.
 <b>PopupMenuFilter</b> ( <a href="#">see page 192</a> )	Specifies which categories of items may be used to form default popup menu.
 <b>UndoLoad</b> ( <a href="#">see page 192</a> )	Indicates that designer is in Undo ( <a href="#">see page 183</a> ) loading state, i.e. in reading previously saved form resource.
 <b>Root</b> ( <a href="#">see page 192</a> )	Root component for TzCustomFormDesigner ( <a href="#">see page 161</a> ).
 <b>RootModified</b> ( <a href="#">see page 192</a> )	Indicates whether the Root ( <a href="#">see page 192</a> ) or its components are modified ( <a href="#">see page 181</a> ).
 <b>SelCount</b> ( <a href="#">see page 193</a> )	Indicates number of selected components.
 <b>Selected</b> ( <a href="#">see page 193</a> )	Indicates whether a particular control is selected.
 <b>SelMarker</b> ( <a href="#">see page 193</a> )	Selection markers manager.
 <b>ShowCaptions</b> ( <a href="#">see page 193</a> )	Specifies whether non-visual component icons captions are visible.
 <b>SnapToGrid</b> ( <a href="#">see page 193</a> )	Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines.
 <b>Target</b> ( <a href="#">see page 193</a> )	Specifies object that is edited by the designer.











## TzFormDesigner Class

<b>TzFormDesigner Class</b>	<b>Description</b>
 <b>Active</b> ( <a href="#">see page 214</a> )	Switches target component between design and run-time modes
 <b>AllowComponents</b> ( <a href="#">see page 214</a> )	Specifies whether nonvisual components will be displayed in design mode
 <b>AutoAlign</b> ( <a href="#">see page 214</a> )	Specifies using of align rulers.
 <b>BDSStyle</b> ( <a href="#">see page 214</a> )	Specifies using of BDS style design environment.
 <b>CaptionFont</b> ( <a href="#">see page 214</a> )	Controls the text attributes of non-visual components captions.
 <b>CloseDisactive</b> ( <a href="#">see page 214</a> )	Specifies if TzCustomFormDesigner automatically deactivates when Target Form is to be closed.
 <b>DesignSurface</b> ( <a href="#">see page 215</a> )	Specifies design surface.
 <b>DisplayControlGrid</b> ( <a href="#">see page 215</a> )	Specifies whether designer has to paint grid over window controls that can accept controls.
 <b>DisplayGrid</b> ( <a href="#">see page 215</a> )	Determines whether dots are drawn on the Target form.
 <b>DragParentLimit</b> ( <a href="#">see page 215</a> )	Specifies whether drag mouse movement should be clipped by parent's client area.
 <b>FlatIcons</b> ( <a href="#">see page 215</a> )	Determines whether the non-visual component icons has a 3D border or not.



























 GridStepX ( <a href="#">see page 215</a> )	Specifies grid step, in pixels, along X-axis
 GridStepY ( <a href="#">see page 216</a> )	Specifies grid step, in pixels, along Y-axis
 GuidelinesStyle ( <a href="#">see page 216</a> )	Specifies guidelines options.
 IgnoreReadErrors ( <a href="#">see page 216</a> )	Specifies whether read errors should be ignored when loading using LoadFromFile and LoadFromStream methods.
 LockControls ( <a href="#">see page 216</a> )	Specifies if user can directly change size and position of controls by mouse.
 LockPublished ( <a href="#">see page 216</a> )	Specifies if editing operation are forbidden by default
 MultiSelect ( <a href="#">see page 216</a> )	Determines whether the user can select more than one control at a time.
 OnActiveChanged ( <a href="#">see page 217</a> )	Occurs when the Active property of the TzCustomFormDesigner changes
 OnCanDelete ( <a href="#">see page 217</a> )	Occurs before deleting selected component.
 OnCanEdit ( <a href="#">see page 217</a> )	Occurs to determine whether Edit method of component editor can be called.
 OnCanInsert ( <a href="#">see page 217</a> )	Occurs before inserting new component.
 OnCanMove ( <a href="#">see page 217</a> )	Occurs before moving selected component.
 OnCanRename ( <a href="#">see page 218</a> )	Occurs before renaming component.
 OnCanResize ( <a href="#">see page 218</a> )	Occurs before resizing selected component.
 OnCanSelect ( <a href="#">see page 218</a> )	Occurs before selecting component.
 OnCreateComponent ( <a href="#">see page 218</a> )	Occurs before new component creation.
 OnCreateFrame ( <a href="#">see page 219</a> )	Occurs when frame is to be inserted on the form.
 OnCreateIcon ( <a href="#">see page 219</a> )	Occurs before creating icon for non-visual component.
 OnCreateMethod ( <a href="#">see page 219</a> )	Occurs when new method name is input in object inspector.
 OnDragDrop ( <a href="#">see page 219</a> )	Occurs when the user drops an object being dragged.
 OnDragOver ( <a href="#">see page 220</a> )	Occurs when the user drags an object over a control.
 OnDrawControl ( <a href="#">see page 220</a> )	Occurs when painting any control on the form. Use this event to draw over control.
 OnExecuteAction ( <a href="#">see page 220</a> )	Occurs when ExecuteAction method is called. Use it to handle standard shared actions for currently active designer.
 OnFormClosed ( <a href="#">see page 220</a> )	Occurs immediately after hiding the Target form .
 OnGetComponentHint ( <a href="#">see page 220</a> )	Occurs when the application is about to display the hint window for the particular component.
 OnGetComponentLocked ( <a href="#">see page 221</a> )	Occurs to determine whether component is locked.
 OnGetMethodNames ( <a href="#">see page 221</a> )	Occurs when method property editor requests designer for possible method names.
 OnGetObjectName ( <a href="#">see page 221</a> )	Occurs at the end of GetObjectName ( <a href="#">see page 177</a> ) method to adjust resulting name.
 OnGetScriptProc ( <a href="#">see page 222</a> )	Occurs when method property editor ask for script procedure name associated with a given property.
 OnHandleControlMessage ( <a href="#">see page 222</a> )	Occurs on any message sent to managed controls.
 OnKeyDown ( <a href="#">see page 222</a> )	Occurs only at design mode when user presses down any key.
 OnKeyPress ( <a href="#">see page 223</a> )	Occurs only at design mode when key pressed.
 OnKeyUp ( <a href="#">see page 223</a> )	Occurs only at design mode when user releases key that has been pressed.
 OnMouseDown ( <a href="#">see page 223</a> )	Occurs only at design mode when user presses mouse button.
 OnMouseMove ( <a href="#">see page 223</a> )	Occurs only at design mode when user moves mouse.
 OnMouseUp ( <a href="#">see page 224</a> )	Occurs only at design mode when user releases mouse button.
 OnNotification ( <a href="#">see page 224</a> )	Occurs when components are added or removed to/from Root object at design mode.
 OnPopUndo ( <a href="#">see page 224</a> )	Occurs when restoring Target from undo buffer.
 OnPushUndo ( <a href="#">see page 224</a> )	Occurs when saving Target to undo buffer.
 OnRenameMethod ( <a href="#">see page 224</a> )	Occurs when name of method is changed in object inspector.
 OnSetNewName ( <a href="#">see page 225</a> )	Occurs when assigning name to newly inserted component (created or pasted).
 OnSetScriptProc ( <a href="#">see page 225</a> )	Occurs when method property editor assigns script procedure to the event.
 OnShowMethod ( <a href="#">see page 225</a> )	Occurs when user double clicks on the procedure in the object inspector.
 OnUpdateAction ( <a href="#">see page 225</a> )	Occurs when UpdateAction method is called. Use it to handle standard shared actions for currently active designer.
 OnValidateMethod ( <a href="#">see page 225</a> )	Occurs to validate method.
 PopupMenu ( <a href="#">see page 226</a> )	Identifies the pop-up menu associated with the Root control of the Designer.
 PopupMenuFilter ( <a href="#">see page 226</a> )	Specifies which categories of items may be used to form default popup menu.
 ReadOnly ( <a href="#">see page 226</a> )	Set Read Only mode.
 SelMarker ( <a href="#">see page 226</a> )	Selection markers manager.
 ShowCaptions ( <a href="#">see page 227</a> )	Specifies whether non-visual component icons captions are visible.
 ShowHints ( <a href="#">see page 227</a> )	Specifies showing of design hints.



 SnapToGrid ( <a href="#">see page 227</a> )	Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines.
 StoreEvents ( <a href="#">see page 227</a> )	Specifies whether designer should process events storage.
 TabOrderIcons ( <a href="#">see page 227</a> )	Properties of tab order icons. These controls are shown over children control when designer is in "Show Tab Order" mode.
 Target ( <a href="#">see page 227</a> )	Specifies object that is edited by the designer.
 TextEditMode ( <a href="#">see page 228</a> )	Sets in-place text editing mode of designer.
 UndoLimit ( <a href="#">see page 229</a> )	Specifies the number of changes that can be undone.

## TBaseDesigner Events







TBaseDesigner Events	Description
 OnActiveChanged ( <a href="#">see page 152</a> )	Occurs when the Active ( <a href="#">see page 151</a> ) property of the TzCustomFormDesigner changes
 OnDragDrop ( <a href="#">see page 152</a> )	Occurs when the user drops an object being dragged.
 OnDragOver ( <a href="#">see page 152</a> )	Occurs when the user drags an object over a control.
 OnHandleControlMessage ( <a href="#">see page 153</a> )	Occurs on any message sent to managed controls.
 OnKeyDown ( <a href="#">see page 153</a> )	Occurs only at design mode when user presses down any key.
 OnKeyPress ( <a href="#">see page 153</a> )	Occurs only at design mode when key pressed.
 OnKeyUp ( <a href="#">see page 154</a> )	Occurs only at design mode when user releases key that has been pressed.
 OnMouseDown ( <a href="#">see page 154</a> )	Occurs only at design mode when user presses mouse button.
 OnMouseMove ( <a href="#">see page 154</a> )	Occurs only at design mode when user moves mouse.
 OnMouseUp ( <a href="#">see page 154</a> )	Occurs only at design mode when user releases mouse button.

## TzCustomFormDesigner Class











TzCustomFormDesigner Class	Description
 OnCanEdit ( <a href="#">see page 194</a> )	Occurs to determine whether Edit ( <a href="#">see page 175</a> ) method of component editor can be called.
 OnDrawControl ( <a href="#">see page 195</a> )	Occurs when painting any control on the form. Use this event to draw over control.
 OnExecuteAction ( <a href="#">see page 195</a> )	Occurs when ExecuteAction ( <a href="#">see page 175</a> ) method is called. Use it to handle standard shared actions for currently active designer.
 OnGetComponentLocked ( <a href="#">see page 195</a> )	Occurs to determine whether component is locked.
 OnCanDelete ( <a href="#">see page 195</a> )	Occurs before deleting selected component.
 OnCanInsert ( <a href="#">see page 195</a> )	Occurs before inserting new component.
 OnGetObjectNames ( <a href="#">see page 195</a> )	Occurs at the end of GetObjectName ( <a href="#">see page 177</a> ) method to adjust resulting name.
 OnCanMove ( <a href="#">see page 196</a> )	Occurs before moving selected component.
 OnCanRename ( <a href="#">see page 196</a> )	Occurs before renaming component.
 OnCanResize ( <a href="#">see page 196</a> )	Occurs before resizing selected component.
 OnPopUndo ( <a href="#">see page 196</a> )	Occurs when restoring Target ( <a href="#">see page 193</a> ) from undo ( <a href="#">see page 183</a> ) buffer.
 OnCanSelect ( <a href="#">see page 196</a> )	Occurs before selecting component.
 OnPushUndo ( <a href="#">see page 197</a> )	Occurs when saving Target ( <a href="#">see page 193</a> ) to undo ( <a href="#">see page 183</a> ) buffer.
 OnCreateComponent ( <a href="#">see page 197</a> )	Occurs before new component creation.
 OnCreateFrame ( <a href="#">see page 197</a> )	Occurs when frame is to be inserted on the form.
 OnSetNewName ( <a href="#">see page 197</a> )	Occurs when assigning name to newly inserted component (created or pasted).
 OnCreateIcon ( <a href="#">see page 197</a> )	Occurs before creating icon for non-visual component.
 OnCreateMethod ( <a href="#">see page 198</a> )	Occurs when new method name is input in object inspector.
 OnFormClosed ( <a href="#">see page 198</a> )	Occurs immediately after hiding the Target ( <a href="#">see page 193</a> ) form .
 OnUpdateAction ( <a href="#">see page 198</a> )	Occurs when UpdateAction ( <a href="#">see page 182</a> ) method is called. Use it to handle standard shared actions for currently active designer.
 OnGetComponentHint ( <a href="#">see page 198</a> )	Occurs when the application is about to display the hint window for the particular component.
 OnGetMethodNames ( <a href="#">see page 198</a> )	Occurs when method property editor requests designer for possible method names.
 OnGetScriptProc ( <a href="#">see page 199</a> )	Occurs when method property editor ask for script procedure name associated with a given property.
 OnNotification ( <a href="#">see page 199</a> )	Occurs when components are added or removed to/from Root ( <a href="#">see page 192</a> ) object at design mode.
 OnRenameMethod ( <a href="#">see page 200</a> )	Occurs when name of method is changed in object inspector.
 OnSetScriptProc ( <a href="#">see page 200</a> )	Occurs when method property editor assigns script procedure to the event.

 OnShowMethod (see page 201)	Occurs when user double clicks on the procedure in the object inspector.
 OnValidateMethod (see page 201)	Occurs to validate method.

























**Legend**

	Method
	protected
	virtual
	Property
	read only
	Event





**TBaseDesigner Events**

TBaseDesigner Events	Description
 OnActiveChanged (see page 152)	Occurs when the Active (see page 151) property of the TzCustomFormDesigner changes
 OnDragDrop (see page 152)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 152)	Occurs when the user drags an object over a control.
 OnHandleControlMessage (see page 153)	Occurs on any message sent to managed controls.
 OnKeyDown (see page 153)	Occurs only at design mode when user presses down any key.
 OnKeyPress (see page 153)	Occurs only at design mode when key pressed.
 OnKeyUp (see page 154)	Occurs only at design mode when user releases key that has been pressed.
 OnMouseDown (see page 154)	Occurs only at design mode when user presses mouse button.
 OnMouseMove (see page 154)	Occurs only at design mode when user moves mouse.
 OnMouseUp (see page 154)	Occurs only at design mode when user releases mouse button.























**TzCustomFormDesigner Class**

TzCustomFormDesigner Class	Description
 OnCanEdit (see page 194)	Occurs to determine whether Edit (see page 175) method of component editor can be called.
 OnDrawControl (see page 195)	Occurs when painting any control on the form. Use this event to draw over control.
 OnExecuteAction (see page 195)	Occurs when ExecuteAction (see page 175) method is called. Use it to handle standard shared actions for currently active designer.
 OnGetComponentLocked (see page 195)	Occurs to determine whether component is locked.
 OnCanDelete (see page 195)	Occurs before deleting selected component.
 OnCanInsert (see page 195)	Occurs before inserting new component.
 OnGetObjectNames (see page 195)	Occurs at the end of GetObjectName (see page 177) method to adjust resulting name.
 OnCanMove (see page 196)	Occurs before moving selected component.
 OnCanRename (see page 196)	Occurs before renaming component.
 OnCanResize (see page 196)	Occurs before resizing selected component.
 OnPopUndo (see page 196)	Occurs when restoring Target (see page 193) from undo (see page 183) buffer.
 OnCanSelect (see page 196)	Occurs before selecting component.
 OnPushUndo (see page 197)	Occurs when saving Target (see page 193) to undo (see page 183) buffer.
 OnCreateComponent (see page 197)	Occurs before new component creation.
 OnCreateFrame (see page 197)	Occurs when frame is to be inserted on the form.
 OnSetNewName (see page 197)	Occurs when assigning name to newly inserted component (created or pasted).
 OnCreateIcon (see page 197)	Occurs before creating icon for non-visual component.
 OnCreateMethod (see page 198)	Occurs when new method name is input in object inspector.
 OnFormClosed (see page 198)	Occurs immediately after hiding the Target (see page 193) form .
 OnUpdateAction (see page 198)	Occurs when UpdateAction (see page 182) method is called. Use it to handle standard shared actions for currently active designer.
 OnGetComponentHint (see page 198)	Occurs when the application is about to display the hint window for the particular component.
 OnGetMethodNames (see page 198)	Occurs when method property editor requests designer for possible method names.
 OnGetScriptProc (see page 199)	Occurs when method property editor ask for script procedure name associated with a given property.
 OnNotification (see page 199)	Occurs when components are added or removed to/from Root (see page 192) object at design mode.




 OnRenameMethod (see page 200)	Occurs when name of method is changed in object inspector.
 OnSetScriptProc (see page 200)	Occurs when method property editor assigns script procedure to the event.
 OnShowMethod (see page 201)	Occurs when user double clicks on the procedure in the object inspector.
 OnValidateMethod (see page 201)	Occurs to validate method.

















## TBaseDesigner Methods

TBaseDesigner Methods	Description
 CanProcessNCMessages (see page 146)	Called to check processing of non client mouse messages.
 Client2Screen (see page 147)	Corrected version of ClientToScreen function to support RTL.
 ClientOrg (see page 147)	Corrected version of ClientOrigin function to support RTL.
 Create (see page 147)	Creates and initializes a TBaseDesigner instance.
 DesignState (see page 147)	Indicates whether component in design or loading state. At design time (in Delphi IDE) and during loading no activation occurs.
 Destroy (see page 147)	Destroys an instance of TBaseDesigner.
 DoObjectHint (see page 147)	Called to show hint for current design state.
 DragDrop (see page 147)	OnDragDrop (see page 152) event dispatcher.
 DragOver (see page 148)	OnDragOver (see page 152) event dispatcher.
 IsRTL (see page 148)	Returns true when control is right-to-left.
 KeyDown (see page 148)	Respond to key press events.
 KeyPress (see page 149)	Respond to keyboard input.
 KeyUp (see page 149)	Respond to released key.
 Loaded (see page 149)	Initializes the component after the form file has been read into memory.
 MouseDown (see page 150)	OnMouseDown (see page 154) event dispatcher.
 MouseMove (see page 150)	OnMouseMove (see page 154) event dispatcher.
 MouseUp (see page 150)	OnMouseUp (see page 154) event dispatcher.
 ProcessMessage (see page 151)	Translates messages of all managed by designer controls.
 ResetHint (see page 151)	Resets hint for another component
 Screen2Client (see page 151)	Corrected version of ScreenToClient function to support RTL.
 SetActive (see page 151)	Set method for Active (see page 151) property.
 ShowHint (see page 151)	Indicates whether hints should be shown for controls in the Designer.










## IClassSelector Interface

IClassSelector Interface	Description
 ClsChanged (see page 511)	Called when selected in component palette component class was changed.
 ClsPalChanged (see page 512)	Called when component palette was changed.




## TzCustomFormDesigner Class

TzCustomFormDesigner Class	Description
 AddCompEditorMenu (see page 170)	Adds menu items to the popup Menu associated with the component editor of selected in designer component. To remove previously added component editor's menu items use ClearCompEditorMenu (see page 171) method.
 CanRedo (see page 170)	Indicates whether the designer contains undone changes that can be repeated.
 CanUndo (see page 171)	Indicates whether the designer contains changes that can be backed out.
 CheckAction (see page 171)	Determines whether design operation specified by the Action parameter can be executed.
 ClearCompEditorMenu (see page 171)	Removes component editor's menu items added by the AddCompEditorMenu (see page 170) method.
 ClearUndo (see page 171)	Clears the undo (see page 183) buffer so that no changes to the Target (see page 193) can be backed out.
 CloseTextEditor (see page 171)	Closes in-place editor.
 DragDrop (see page 171)	OnDragDrop event dispatcher.
 DragOver (see page 171)	OnDragOver event dispatcher.
 EditAction (see page 172)	Executes designer operation specified by the Action parameter.
 AlignSelected (see page 172)	Performs alignment operation on selected components.
 AlignToGrid (see page 172)	Aligns selected components to the closest grid point.
 BringToFront (see page 172)	Moves a selected component in front of all other components on the form. This is called changing the component's z-order.
 BuildLocalMenu (see page 172)	Creates default popup menu.
 CancelDrag (see page 173)	Cancels current drag&drop design operation.
 CanDelete (see page 173)	Check if component can be deleted.








◆ GetEditState (see page 173)	Returns set of possible designer operations.
◆ CanInsert (see page 173)	Check if component can be inserted.
◆ CanMove (see page 173)	Check if component can be moved.
◆ CanPaste (see page 173)	Checks clipboard for components saved to it.
◆ CanRename (see page 173)	Check if component can be renamed.
◆ CanResize (see page 174)	Check if component can be resized.
◆ CanSelect (see page 174)	Check if component can be selected.
◆ ClearSelection (see page 174)	Resets selection.
◆ CopySelection (see page 174)	
◆ Create (see page 174)	Creates and initializes a TzCustomFormDesigner instance.
◆ CutSelection (see page 174)	Cuts selected components to clipboard.
◆ DeleteSelection (see page 175)	Deletes the selected component or components
◆ Destroy (see page 175)	Destroys an instance of TzCustomFormDesigner.
◆ DoObjectHint (see page 175)	Shows hint current design state.
◆ ExecuteAction (see page 175)	Invokes an action with the component as its target.
◆ DragDraw (see page 175)	Provides drag-and-draw graphic operation.
◆ Edit (see page 175)	Displays the component editor for the specified component.
◆ EndDrag (see page 176)	Provides drag-and-draw graphic operation.
◆ FlipChildren (see page 176)	Allows to reverse the layout of components in the current form to a right-to-left mirror image.
◆ GetCompObj (see page 176)	Returns component
◆ GetComponent (see page 176)	Returns the component with the name passed as a parameter.
◆ GetComponentName (see page 176)	Returns the name of the component passed as its parameter.
◆ GetComponentNames (see page 177)	Executes a callback for every component that can be assigned a property of a specified type.
◆ GetControlAt (see page 177)	Looks for control at specified point on the window prn.
◆ GetMethodName (see page 177)	Returns the name of a specified event handler.
◆ GetNewName (see page 177)	Returns new generated name for particular class of the component.
◆ GetObjectName (see page 177)	Returns object name.
◆ LoadFromFile (see page 177)	Load Root (see page 192) component from file. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.
◆ LoadFromStream (see page 178)	Load Root (see page 192) component from stream. Inline events from resource file are stored in Events (see page 189) property if property StoreEvents (see page 191) is True. If IgnoreReadErrors (see page 191) is True - all read errors are ignored.
◆ GetRoot (see page 178)	Returns the current entity being edited by the form designer.
◆ GetRootClassName (see page 178)	Returns the class name for the root component.
◆ GetScriptEvent (see page 178)	Returns script procedure name assigned to the event.
◆ GetScrollRanges (see page 178)	Returns the size of the logical designer window.
◆ GetSelections (see page 178)	Fills a list with all selected components on the current root object.
◆ Redo (see page 179)	Call Redo to repeat last undone operation.
◆ GetShiftState (see page 179)	Returns the current state of the Shift, Alt, and Ctrl keys.
◆ Intf_Notification (see page 179)	Allows the designer to respond when a notification (see page 183) is sent to the form.
◆ IsComponentHidden (see page 179)	Indicates whether a component does not appear directly in the form designer.
◆ IsLocked (see page 179)	Specifies whether component is locked and can not be edited in designer.
◆ SaveToFile (see page 179)	Saves Root (see page 192) component to file FileName with events information. AsText specifies format of the file: text or binary.
◆ IsDesignMsg (see page 180)	Determines when the designer should handle a Windows message.
◆ SaveToStream (see page 180)	Save Root (see page 192) component to stream with events information. AsText specifies storage format: text or binary.
◆ IsProtected (see page 180)	Indicates whether component is protected and can not be changed by designer.
◆ IsRootSelected (see page 180)	Returns True if entire Root (see page 192) is selected
◆ IsSourceReadOnly (see page 180)	Indicates whether the source file for the component being designed is read-only.
◆ SelectedComponentsCount (see page 180)	Returns number of selected components excluding Root (see page 192) and non-component objects.
◆ KeyDown (see page 180)	Respond to key press events.
◆ KeyPress (see page 181)	Respond to keyboard input.
◆ KeyUp (see page 181)	Respond to released key.

 MethodExists (see page 181)	Indicates whether an event handler with a specified name already exists.
 Modified (see page 181)	Notifies property and component editors when a change is made to a component.
 ShowTabOrder (see page 182)	Shows tab order icons over children controls of the selected control. Click on the children controls changes their tab order. To exit "Show Tab Icons" mode click on any not child control or press ESCAPE key.
  MouseDown (see page 182)	Generates an OnMouseDown event.
  MouseMove (see page 182)	Generates an OnMouseMove event.
  MouseUp (see page 182)	Generates an OnMouseUp event.
 Navigate (see page 182)	Performs navigation between control using keyboard
  UpdateAction (see page 182)	Updates an action component to reflect the current state of the component.
 NoSelection (see page 183)	Deselects all components in the form designer.
 Undo (see page 183)	Backs out last change in the undo buffer.
  Notification (see page 183)	Allows the designer to respond when a notification is sent to the form.
  NotifySelChanged (see page 183)	Notifies active designer about changing selection list
  PaintControl (see page 184)	Called for each WM_PAINT message to perform specific painting over control.
 PaintGrid (see page 184)	Paints the alignment grid on the form's canvas.
 PasteSelection (see page 184)	Pastes the contents of the clipboard into the selected component or components.
  ReadComp (see page 184)	Callback procedure for TReader.ReadComponents
 RenameMethod (see page 184)	Renames an existing event handler.
 Scale (see page 185)	Scale all controls using defined ratio
 SelectAll (see page 185)	Selects all components.
 SelectComponent (see page 185)	Replaces the current set of selected components by a single specified object.
 SelectedComponent (see page 185)	Returns selected component if selection consist of single component; Root (see page 192) component, if there is no selection; nil if selection consists of several components.
  SelectionChanged (see page 185)	This method call after selection has been changed.
 SelectObj (see page 185)	Selects object passed as Instance parameter.
 SelectRect (see page 185)	Selects all controls on the Prn control.
 SendToBack (see page 186)	Moves a selected component behind all other components on the form. This is called changing the component's z-order.
  SetPasteName (see page 186)	Setting new name for pasting component
 SetScriptEvent (see page 186)	Assigns script procedure with particular event of the Instance.
 SetSelections (see page 186)	Changes the currently selected set of components.
 ShowMethod (see page 186)	Activates the code editor with the input cursor in a specified event handler.
 ShowPopupMenu (see page 186)	Show designer popup menu at the specified screen position.
 SizeSelected (see page 186)	Perform size operation to selected components
 StartDrag (see page 187)	Makes start settings before dragging operation
 UniqueName (see page 187)	Generates a unique name from a specified base string.
 UpdateComplcons (see page 187)	Updates positions of component icons.
 ValidateMethod (see page 187)	Determines whether method MAddr of object ARoot is valid method, i.e. it may be assigned to the event.

## TBaseDesigner Properties

TBaseDesigner Properties	Description
 Active (see page 151)	Switches target component between design and run-time modes
 HintObject (see page 152)	Specifies object for which hint was activated.
 ShowHints (see page 152)	Specifies showing of design hints.


















## TzCustomFormDesigner Class




















































TzCustomFormDesigner Class	Description
 AllowComponents (see page 187)	Specifies whether nonvisual components will be displayed in design mode
 AutoAlign (see page 187)	Specifies using of align rulers.
 BDSStyle (see page 188)	Specifies using of BDS style design environment.
 CaptionFont (see page 188)	Controls the text attributes of non-visual components captions.
 CloseDisactive (see page 188)	Specifies if TzCustomFormDesigner (see page 161) automatically deactivates when Target (see page 193) Form (see page 189) is to be closed.
 ContainerWindow (see page 188)	Determines generic container for any type of Target (see page 193) components
 DesignSurface (see page 188)	Specifies design surface.



 DisplayControlGrid (see page 189)	Specifies whether designer has to paint grid over window controls that can accept controls.
 DisplayGrid (see page 189)	Determines whether dots are drawn on the Target (see page 193) form.
 FlatIcons (see page 189)	Determines whether the non-visual component icons has a 3D border or not.
 DragParentLimit (see page 189)	Specifies whether drag mouse movement should be clipped by parent's client area.
 Form (see page 189)	Provides access to designed form as TCustomForm type.
 Events (see page 189)	Storage of assigned events.
 GridStepX (see page 190)	Specifies grid step, in pixels, along X-axis
 GridStepY (see page 190)	Specifies grid step, in pixels, along Y-axis
 LockControls (see page 190)	Specifies if user can directly change size and position of controls by mouse.
 LockPublished (see page 190)	Specifies if editing operation are forbidden by default
 MultiSelect (see page 190)	Determines whether the user can select more than one control at a time.
 Groups (see page 190)	Stores groups information.
 GuidelinesStyle (see page 191)	Specifies guidelines options.
 IgnoreReadErrors (see page 191)	Specifies whether read errors should be ignored when loading using LoadFromFile (see page 177) and LoadFromStream (see page 178) methods.
 ReadOnly (see page 191)	Set Read Only mode.
 StoreEvents (see page 191)	Specifies whether designer should process events storage.
 TabOrderIcons (see page 191)	Properties of tab order icons. These controls are shown over children control when designer is in "Show Tab Order" mode.
 TextEditMode (see page 191)	Sets in-place text editing mode of designer.
 PopupMenu (see page 191)	Identifies the pop-up menu associated with the Root (see page 192) control of the Designer.
 UndoLimit (see page 192)	Specifies the number of changes that can be undone.
 PopupMenuFilter (see page 192)	Specifies which categories of items may be used to form default popup menu.
 UndoLoad (see page 192)	Indicates that designer is in Undo (see page 183) loading state, i.e. in reading previously saved form resource.
 Root (see page 192)	Root component for TzCustomFormDesigner (see page 161).
 RootModified (see page 192)	Indicates whether the Root (see page 192) or its components are modified (see page 181).
 SelCount (see page 193)	Indicates number of selected components.
 Selected (see page 193)	Indicates whether a particular control is selected.
 SelMarker (see page 193)	Selection markers manager.
 ShowCaptions (see page 193)	Specifies whether non-visual component icons captions are visible.
 SnapToGrid (see page 193)	Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines.
 Target (see page 193)	Specifies object that is edited by the designer.

## TzFormDesigner Class

TzFormDesigner Class	Description
 Active (see page 214)	Switches target component between design and run-time modes
 AllowComponents (see page 214)	Specifies whether nonvisual components will be displayed in design mode
 AutoAlign (see page 214)	Specifies using of align rulers.
 BDSStyle (see page 214)	Specifies using of BDS style design environment.
 CaptionFont (see page 214)	Controls the text attributes of non-visual components captions.
 CloseDisactive (see page 214)	Specifies if TzCustomFormDesigner automatically deactivates when Target Form is to be closed.
 DesignSurface (see page 215)	Specifies design surface.
 DisplayControlGrid (see page 215)	Specifies whether designer has to paint grid over window controls that can accept controls.
 DisplayGrid (see page 215)	Determines whether dots are drawn on the Target form.
 DragParentLimit (see page 215)	Specifies whether drag mouse movement should be clipped by parent's client area.
 FlatIcons (see page 215)	Determines whether the non-visual component icons has a 3D border or not.
 GridStepX (see page 215)	Specifies grid step, in pixels, along X-axis
 GridStepY (see page 216)	Specifies grid step, in pixels, along Y-axis
 GuidelinesStyle (see page 216)	Specifies guidelines options.
 IgnoreReadErrors (see page 216)	Specifies whether read errors should be ignored when loading using LoadFromFile and LoadFromStream methods.
 LockControls (see page 216)	Specifies if user can directly change size and position of controls by mouse.
 LockPublished (see page 216)	Specifies if editing operation are forbidden by default

 MultiSelect (see page 216)	Determines whether the user can select more than one control at a time.
 OnActiveChanged (see page 217)	Occurs when the Active property of the TzCustomFormDesigner changes
 OnCanDelete (see page 217)	Occurs before deleting selected component.
 OnCanEdit (see page 217)	Occurs to determine whether Edit method of component editor can be called.
 OnCanInsert (see page 217)	Occurs before inserting new component.
 OnCanMove (see page 217)	Occurs before moving selected component.
 OnCanRename (see page 218)	Occurs before renaming component.
 OnCanResize (see page 218)	Occurs before resizing selected component.
 OnCanSelect (see page 218)	Occurs before selecting component.
 OnCreateComponent (see page 218)	Occurs before new component creation.
 OnCreateFrame (see page 219)	Occurs when frame is to be inserted on the form.
 OnCreateIcon (see page 219)	Occurs before creating icon for non-visual component.
 OnCreateMethod (see page 219)	Occurs when new method name is input in object inspector.
 OnDragDrop (see page 219)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 220)	Occurs when the user drags an object over a control.
 OnDrawControl (see page 220)	Occurs when painting any control on the form. Use this event to draw over control.
 OnExecuteAction (see page 220)	Occurs when ExecuteAction method is called. Use it to handle standard shared actions for currently active designer.
 OnFormClosed (see page 220)	Occurs immediately after hiding the Target form .
 OnGetComponentHint (see page 220)	Occurs when the application is about to display the hint window for the particular component.
 OnGetComponentLocked (see page 221)	Occurs to determine whether component is locked.
 OnGetMethodNames (see page 221)	Occurs when method property editor requests designer for possible method names.
 OnGetObjectName (see page 221)	Occurs at the end of GetObjectName (see page 177) method to adjust resulting name.
 OnGetScriptProc (see page 222)	Occurs when method property editor ask for script procedure name associated with a given property.
 OnHandleControlMessage (see page 222)	Occurs on any message sent to managed controls.
 OnKeyDown (see page 222)	Occurs only at design mode when user presses down any key.
 OnKeyPress (see page 223)	Occurs only at design mode when key pressed.
 OnKeyUp (see page 223)	Occurs only at design mode when user releases key that has been pressed.
 OnMouseDown (see page 223)	Occurs only at design mode when user presses mouse button.
 OnMouseMove (see page 223)	Occurs only at design mode when user moves mouse.
 OnMouseUp (see page 224)	Occurs only at design mode when user releases mouse button.
 OnNotification (see page 224)	Occurs when components are added or removed to/from Root object at design mode.
 OnPopUndo (see page 224)	Occurs when restoring Target from undo buffer.
 OnPushUndo (see page 224)	Occurs when saving Target to undo buffer.
 OnRenameMethod (see page 224)	Occurs when name of method is changed in object inspector.
 OnSetNewName (see page 225)	Occurs when assigning name to newly inserted component (created or pasted).
 OnSetScriptProc (see page 225)	Occurs when method property editor assigns script procedure to the event.
 OnShowMethod (see page 225)	Occurs when user double clicks on the procedure in the object inspector.
 OnUpdateAction (see page 225)	Occurs when UpdateAction method is called. Use it to handle standard shared actions for currently active designer.
 OnValidateMethod (see page 225)	Occurs to validate method.
 PopupMenu (see page 226)	Identifies the pop-up menu associated with the Root control of the Designer.
 PopupMenuFilter (see page 226)	Specifies which categories of items may be used to form default popup menu.
 ReadOnly (see page 226)	Set Read Only mode.
 SelMarker (see page 226)	Selection markers manager.
 ShowCaptions (see page 227)	Specifies whether non-visual component icons captions are visible.
 ShowHints (see page 227)	Specifies showing of design hints.
 SnapToGrid (see page 227)	Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines.
 StoreEvents (see page 227)	Specifies whether designer should process events storage.
 TabOrderIcons (see page 227)	Properties of tab order icons. These controls are shown over children control when designer is in "Show Tab Order" mode.
 Target (see page 227)	Specifies object that is edited by the designer.
 TextEditMode (see page 228)	Sets in-place text editing mode of designer.
 UndoLimit (see page 229)	Specifies the number of changes that can be undone.

## 1.15.1.4.1 TzFormDesigner Properties

### 1.15.1.4.1.1 TzFormDesigner.Active Property

Switches target component between design and run-time modes

```
property Active: Boolean;
```

#### Description

Active is one of main properties of TzCustomFormDesigner.

It switches target component (it must be TWinControl descendant) between design and run-time mode.

When in design mode all of parents controls are in design mode too so user can manipulate all of this properties.

Manipulating with nonvisual components depends on AllowComponents property.

### 1.15.1.4.1.2 TzFormDesigner.AllowComponents Property

Specifies whether nonvisual components will be displayed in design mode

```
property AllowComponents: Boolean;
```

#### Description

Nonvisual components (not derived from TControl) are normally invisible in run-time.

AllowComponents specifies whether those components will be displayed in design mode.

### 1.15.1.4.1.3 TzFormDesigner.AutoAlign Property

Specifies using of align rulers.

```
property AutoAlign: Boolean;
```

#### Description

Align rulers are auxiliary lines to nearest controls. They allows easily aligning moved/resized control to other controls.

### 1.15.1.4.1.4 TzFormDesigner.BDSStyle Property

Specifies using of BDS style design environment.

```
property BDSStyle: Boolean;
```

#### Description

When BDSStyle is True moving and resizing of controls are performed as in BDS, i.e. control is visible during operation, otherwise old designer operations are used - when only frames are drawn during moving or resizing.

### 1.15.1.4.1.5 TzFormDesigner.CaptionFont Property

Controls the text attributes of non-visual components captions.

```
property CaptionFont: TFont;
```

#### Description

This property controls the text attributes of non-visual components captions when ShowCaptions is True.

### 1.15.1.4.1.6 TzFormDesigner.CloseDisactive Property

Specifies if TzCustomFormDesigner automatically deactivates when Target Form is to be closed.

```
property CloseDisactive: Boolean;
```

**Description**

Set this property to True to force TzCustomFormDesigner automatically deactivates when Target Form is to be closed.

**1.15.1.4.1.7 TzFormDesigner.DesignSurface Property**

Specifies design surface.

```
property DesignSurface: TDesignSurface;
```

**Description**

When design surface, control of TDesignSurface class, is assigned form is placed on it. Form does not activated, when user clicks on the form design surface gets focus.

Design surface is usual control which can not be placed on any container - form, tab sheet, panel, etc.

This allows organizing of multiple documents applications where each form is placed on separate tab sheet.

Also design surface allows hiding of form caption and border by setting TDesignSurface.HideFormBorders property.

Using design surface you may implement BDS like design environment.

**1.15.1.4.1.8 TzFormDesigner.DisplayControlGrid Property**

Specifies whether designer has to paint grid over window controls that can accept controls.

```
property DisplayControlGrid: Boolean;
```

**Description**

Set this property to True to display design grid over window controls that can accept controls, for example, over tab sheet or panel.

**1.15.1.4.1.9 TzFormDesigner.DisplayGrid Property**

Determines whether dots are drawn on the Target form.

```
property DisplayGrid: Boolean;
```

**Description**

Set DisplayGrid to True for dots representing as grid on the Target Form.

Dots will be shown only if Target is a TCustomForm descendant.

**1.15.1.4.1.10 TzFormDesigner.DragParentLimit Property**

Specifies whether drag mouse movement should be clipped by parent's client area.

```
property DragParentLimit: Boolean;
```

**1.15.1.4.1.11 TzFormDesigner.FlatIcons Property**

Determines whether the non-visual component icons has a 3D border or not.

```
property FlatIcons: Boolean;
```

**Description**

Set FlatIcons to True to remove the 3D border around the non-visual component icons.

**1.15.1.4.1.12 TzFormDesigner.GridStepX Property**

Specifies grid step, in pixels, along X-axis

```
property GridStepX: integer;
```

**Description**

Use this property to read or change grid step along X-axis.

**1.15.1.4.1.13 TzFormDesigner.GridStepY Property**

Specifies grid step, in pixels, along Y-axis

```
property GridStepY: integer;
```

**Description**

Use this property to read or change grid step along Y-axis.

**1.15.1.4.1.14 TzFormDesigner.GuidelinesStyle Property**

Specifies guidelines options.

```
property GuidelinesStyle: TGuidelinesStyles;
```

**1.15.1.4.1.15 TzFormDesigner.IgnoreReadErrors Property**

Specifies whether read errors should be ignored when loading using LoadFromFile and LoadFromStream methods.

```
property IgnoreReadErrors: Boolean;
```

**1.15.1.4.1.16 TzFormDesigner.LockControls Property**

Specifies if user can directly change size and position of controls by mouse.

```
property LockControls: Boolean;
```

**Description**

Set this property to False to allow user changing controls position and size only through Object Inspector (TObjectInspector).

**1.15.1.4.1.17 TzFormDesigner.LockPublished Property**

Specifies if editing operation are forbidden by default

```
property LockPublished: Boolean;
```

**Description**

Set this property to True to programmatically control what kind of editing will be allowed.

User can adjust reaction on editing request in event handlers.

It affects on events

- OnCanDelete
- OnCanSelect
- OnCanResize
- OnCanMove
- OnCanRename

**1.15.1.4.1.18 TzFormDesigner.MultiSelect Property**

Determines whether the user can select more than one control at a time.

```
property MultiSelect: Boolean;
```

**Description**

Set MultiSelect to True to allow the user to select multiple controls. If MultiSelect if False, multiple controls cannot be selected at the same time.

#### 1.15.1.4.1.19 TzFormDesigner.OnActiveChanged Property

Occurs when the Active property of the TzCustomFormDesigner changes

**property** OnActiveChanged: TNotifyEvent;

##### Description

Write an OnActiveChange event handler to take specific action immediately after the TzCustomFormDesigner changes its Active property. For example user can prohibit saving or loading form being designed while Active = True.

The Sender parameter is the object whose event handler is called.

#### 1.15.1.4.1.20 TzFormDesigner.OnCanDelete Property

Occurs before deleting selected component.

**property** OnCanDelete: TComponentEvent;

##### Description

Write an OnCanDelete event handler to provide custom action. You may forbid deleting for example.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being deleted

The Accept parameter determines whether a component is allowed to delete.

#### 1.15.1.4.1.21 TzFormDesigner.OnCanEdit Property

Occurs to determine whether Edit method of component editor can be called.

**property** OnCanEdit: TComponentEvent;

##### Description

Write OnCanEdit event handler to disable component editor call when user double clicks component.

#### 1.15.1.4.1.22 TzFormDesigner.OnCanInsert Property

Occurs before inserting new component.

**property** OnCanInsert: TComponentEvent;

##### Description

Write an OnCanInsert event handler to provide custom action. You may forbid inserting this new component.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being inserted

The Accept parameter determines whether a component is allowed to insert.

#### 1.15.1.4.1.23 TzFormDesigner.OnCanMove Property

Occurs before moving selected component.

**property** OnCanMove: TComponentEvent;

##### Description

Write an OnCanMove event handler to provide custom action. You may forbid moving this component.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being moved

The Accept parameter determines whether a component is allowed to move.

#### 1.15.1.4.1.24 TzFormDesigner.OnCanRename Property

Occurs before renaming component.

```
property OnCanRename: TRenameEvent;
```

##### Description

Write an OnCanRename event handler to provide custom action. You may forbid renaming this component or set another name.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being renamed

The NewName parameter is the suggested name

The Accept parameter determines whether a component is allowed to rename.

#### 1.15.1.4.1.25 TzFormDesigner.OnCanResize Property

Occurs before resizing selected component.

```
property OnCanResize: TComponentEvent;
```

##### Description

Write an OnCanResize event handler to provide custom action. You may forbid resizing this component.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being resized

The Accept parameter determines whether a component is allowed to resize.

#### 1.15.1.4.1.26 TzFormDesigner.OnCanSelect Property

Occurs before selecting component.

```
property OnCanSelect: TComponentEvent;
```

##### Description

Write an OnCanSelect event handler to provide custom action. You may forbid selecting this component.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being selected

The Accept parameter determines whether a component is allowed to select.

#### 1.15.1.4.1.27 TzFormDesigner.OnCreateComponent Property

Occurs before new component creation.

```
property OnCreateComponent: TCreateComponentEvent;
```

**Description**

Write this handler to customize component creation.

AClass - class of inserted component;

AOwner - owner of the component (root)

Instance - reference to component to be created.

**Note:** if you create event handler you should create component in it. Default component creation will be skipped. This allows to restrict component creation.

**1.15.1.4.1.28 TzFormDesigner.OnCreateFrame Property**

Occurs when frame is to be inserted on the form.

```
property OnCreateFrame: TCreateFrameEvent;
```

**Description**

By default PackageMng.CreateFrame method is called. This method shows select frame dialog and creates an instance of frame class. Using this event you may change default behavior.

**1.15.1.4.1.29 TzFormDesigner.OnCreateIcon Property**

Occurs before creating icon for non-visual component.

```
property OnCreateIcon: TCreateIconEvent;
```

**Description**

Write an OnCreateIcon event handler to provide custom action. You may forbid component icon creation, so that component will not be visible on the designed Target, but will be accessible in object inspector.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component icon is being created for

The AllowCreate parameter determines whether an icon will be created.

**1.15.1.4.1.30 TzFormDesigner.OnCreateMethod Property**

Occurs when new method name is input in object inspector.

```
property OnCreateMethod: TCreateMethodEvent;
```

**Description**

Write this event handler to specify method reference when there is no registered method with the specified Name.

Use TypeData to validate possible assignment, i.e. do not assign to Method reference to procedure with another type. This may cause system halt.

Note: This event is intended for only in-code implemented methods. If you are working with script procedures use OnSetScriptProc event.

**1.15.1.4.1.31 TzFormDesigner.OnDragDrop Property**

Occurs when the user drops an object being dragged.

```
property OnDragDrop: TDragDropEvent;
```



**Description**

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control

**1.15.1.4.1.32 TzFormDesigner.OnDragOver Property**

Occurs when the user drags an object over a control.

**property** OnDragOver: TDragOverEvent;

**Description**

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop target, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

Note: Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

**1.15.1.4.1.33 TzFormDesigner.OnDrawControl Property**

Occurs when painting any control on the form. Use this event to draw over control.

**property** OnDrawControl: TDrawControlEvent;

**1.15.1.4.1.34 TzFormDesigner.OnExecuteAction Property**

Occurs when ExecuteAction method is called. Use it to handle standard shared actions for currently active designer.

**property** OnExecuteAction: THandleActionEvent;

**1.15.1.4.1.35 TzFormDesigner.OnFormClosed Property**

Occurs immediately after hiding the Target form .

**property** OnFormClosed: TNotifyEvent;

**Description**

Write an OnFormClosed event handler to provide custom action when designed form is closed.

The Sender parameter is the object whose event handler is called.

**1.15.1.4.1.36 TzFormDesigner.OnGetComponentHint Property**

Occurs when the application is about to display the hint window for the particular component.

**property** OnGetComponentHint: TGetComponentHintEvent;

**Description**

Write an OnGetComponentHint event handler to provide custom action. You may change hint or even forbid to display it.

The Sender parameter is the object whose event handler is called.

The Component parameter is the component that is being selected

The Hint parameter is the hint string that user can change

The ShowHint parameter determines whether a hint will be displayed.

**1.15.1.4.1.37 TzFormDesigner.OnGetComponentLocked Property**

Occurs to determine whether component is locked.

```
property OnGetComponentLocked: TComponentEvent;
```

**Description**

Write OnGetComponentLocked event handler to specify which components are locked, i.e can be edited in designer.

**1.15.1.4.1.38 TzFormDesigner.OnGetMethodNames Property**

Occurs when method property editor requests designer for possible method names.

```
property OnGetMethodNames: TGetMethodNamesEvent;
```

**Description**

Write this handler to specify possible procedure names that can be assigned to the selected in object inspector event.

Use TypeData to filter possible procedures, for example, by procedure parameters. Call Proc for each possible procedure.

This event is intended for integration with external script engines.

**See Also**

OnGetScriptProc, OnSetScriptProc, OnShowMethod, OnRenameMethod

**Example**

This example shows using of the event. EControl Syntax Editor SDK is used as source code analyzer

```
procedure TForm4.zFormDesigner1GetMethodNames(Sender: TObject;
  TypeData: PTypeData; Proc: TGetStrProc);
var i: integer;
    R: TTagBlockCondition;
begin
  with CodeEditor.SyntObj do
    begin
      // Looking for all text ranges with rule "function"
      R := TTagBlockCondition(Owner.BlockRules.ItemByName('function'));
      if R <> nil then
        for i := 0 to RangeCount - 1 do
          if (Ranges[i].Rule = R) then
            Proc(TagStr[Ranges[i].StartIdx + 1]); // Adds function to procedures list
        end;
    end;
end;
```

**1.15.1.4.1.39 TzFormDesigner.OnGetObjectName Property**

Occurs at the end of GetObjectName (see page 177) method to adjust resulting name.

```
property OnGetObjectName: TGetObjNameEvent;
```

**Description**

This event may be helpful if visible control is only proxy of real object.

#### 1.15.1.4.1.40 TzFormDesigner.OnGetScriptProc Property

Occurs when method property editor ask for script procedure name associated with a given property.

```
property OnGetScriptProc: TGetScriptProcEvent;
```

##### Description

Write this event handler to return procedure name associated with property *pInfo^.Name* of object *Instance*.

##### See Also

OnGetMethodNames, OnSetScriptProc, OnShowMethod, OnRenameMethod

##### Example

In this sample associations of script procedures and properties are stores in TStrings object (property Items in list box EventsList) with items - <ObjectName>.<PropertyName>=<ScriptProcedure>

```
procedure TForm4.zFormDesigner1GetScriptProc(Sender, Instance: TObject;  
  pInfo: PPropInfo; var ProcName: String);  
begin  
  ProcName := '';  
  if Instance is TComponent then  
    if zFormDesigner1.Root = Instance then  
      ProcName := EventsList.Items.Values[pInfo^.Name]  
    else  
      ProcName := EventsList.Items.Values[(Instance as TComponent).Name + '.' +  
      pInfo^.Name];  
end;
```

#### 1.15.1.4.1.41 TzFormDesigner.OnHandleControlMessage Property

Occurs on any message sent to managed controls.

```
property OnHandleControlMessage: THandleControlMessage;
```

##### Description

Write this event handler to process messages.

Sender - designer component;

Control - control to which message was sent;

Message - message;

Handled - handling flag. Set this flag to True to abort following message processing.

#### 1.15.1.4.1.42 TzFormDesigner.OnKeyDown Property

Occurs only at design mode when user presses down any key.

```
property OnKeyDown: TKeyEvent;
```

##### Description

It is the same as standard TWinControl.OnKeyDown event.

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys, and pressed mouse buttons.

See TWinControl.OnKeyDown for details

#### 1.15.1.4.1.43 TzFormDesigner.OnKeyPress Property

Occurs only at design mode when key pressed.

**property** OnKeyPress: TKeyPressEvent;

##### Description

It is the same as standard TWinControl.OnKeyPress event.

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

See TWinControl.OnKeyPress for details

#### 1.15.1.4.1.44 TzFormDesigner.OnKeyUp Property

Occurs only at design mode when user releases key that has been pressed.

**property** OnKeyUp: TKeyEvent;

##### Description

It is the same as standard TWinControl.OnKeyUp event.

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

See TWinControl.OnKeyUp for details

#### 1.15.1.4.1.45 TzFormDesigner.OnMouseDown Property

Occurs only at design mode when user presses mouse button.

**property** OnMouseDown: TMouseEvent;

##### Description

It is the same as standard TControl.OnMouseDown event.

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

See TControl.OnMouseDown for details.

#### 1.15.1.4.1.46 TzFormDesigner.OnMouseMove Property

Occurs only at design mode when user moves mouse.

**property** OnMouseMove: TMouseMoveEvent;

##### Description

It is the same as standard TControl.OnMouseMove event.

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

See TControl.OnMouseMove for details.

#### 1.15.1.4.1.47 TzFormDesigner.OnMouseUp Property

Occurs only at design mode when user releases mouse button.

**property** OnMouseUp: TMouseEvent;

##### Description

It is the same as standard TControl.OnMouseUp event.

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

See TControl.OnMouseUp for details.

#### 1.15.1.4.1.48 TzFormDesigner.OnNotification Property

Occurs when components are added or removed to/from Root object at design mode.

**property** OnNotification: TNotificationEvent;

##### Description

Write an OnNotification event handler to provide custom action when components are added or removed to/from Root object at design mode.

The Sender parameter is the object whose event handler is called.

The AnObject parameter is the TPersistent object that is being operated

The Operation parameter specifies kind of operation applied to object

- opInsert - object inserted
- opRemove - object removed.

#### 1.15.1.4.1.49 TzFormDesigner.OnPopUndo Property

Occurs when restoring Target from undo buffer.

**property** OnPopUndo: TUndoRecEvent;

#### 1.15.1.4.1.50 TzFormDesigner.OnPushUndo Property

Occurs when saving Target to undo buffer.

**property** OnPushUndo: TUndoRecEvent;

#### 1.15.1.4.1.51 TzFormDesigner.OnRenameMethod Property

Occurs when name of method is changed in object inspector.

**property** OnRenameMethod: TRenameMethodEvent;

##### Description

Write this event handler to change script procedure name in script code. You should find script procedure CurName and change its name to NewName.

##### See Also

OnGetMethodNames, OnSetScriptProc, OnShowMethod, OnGetScriptProc

#### 1.15.1.4.1.52 TzFormDesigner.OnSetNewName Property

Occurs when assigning name to newly inserted component (created or pasted).

```
property OnSetNewName: TSetNameEvent;
```

#### 1.15.1.4.1.53 TzFormDesigner.OnSetScriptProc Property

Occurs when method property editor assigns script procedure to the event.

```
property OnSetScriptProc: TSetScriptProcEvent;
```

##### Description

Write this event handler to save association between event property end script procedure.

##### See Also

OnGetMethodNames, OnRenameMethod, OnShowMethod, OnGetScriptProc

##### Example

In this sample associations of script procedures and properties are stores in TStrings object (property Items in list box EventsList) with items - <ObjectName>.<PropertyName>=<ScriptProcedure>

```
procedure TForm4.zFormDesigner1SetScriptProc(Sender, Instance: TObject;  
  pInfo: PPropInfo; const EventProc: String);  
var idx: integer;  
    pn: string;  
begin  
  if Instance is TComponent then  
    begin  
      // event name for root object is without object name  
      if zFormDesigner1.Root = Instance then  
        pn := PInfo^.Name  
      else  
        pn := (Instance as TComponent).Name + '.' + PInfo^.Name;  
      // Delete previous association  
      idx := EventsList.Items.IndexOfName(pn);  
      if idx <> -1 then  
        EventsList.Items.Delete(idx);  
      // Add new association  
      if EventProc <> '' then  
        begin  
          // Saving associating  
          EventsList.Items.Add(pn + '=' + EventProc);  
          // Creating event handler text body  
          CreateMethod(EventProc, Instance, pInfo);  
        end;  
      end;  
    end;  
end;
```

#### 1.15.1.4.1.54 TzFormDesigner.OnShowMethod Property

Occurs when user double clicks on the procedure in the object inspector.

```
property OnShowMethod: TShowMethodEvent;
```

##### Description

Write this handler to highlight script procedure MethodName in script code text.

#### 1.15.1.4.1.55 TzFormDesigner.OnUpdateAction Property

Occurs when UpdateAction method is called. Use it to handle standard shared actions for currently active designer.

```
property OnUpdateAction: THandleActionEvent;
```

#### 1.15.1.4.1.56 TzFormDesigner.OnValidateMethod Property

Occurs to validate method.

```
property OnValidateMethod: TValidateMethodEvent;
```

#### Description

Write an OnValidateMethod event handler to provide custom validation of the particular method.

The Sender parameter is the object whose event handler is called.

The TypeData parameter is the type of method

The ARoot parameter is the owner of this method

The MethAddr parameter is the address of this method

The MethodName parameter is the name of method

The Accept parameter determines whether this method is allowed to add to list of methods.

### 1.15.1.4.1.57 TzFormDesigner.PopupMenu Property

Identifies the pop-up menu associated with the Root control of the Designer.

```
property PopupMenu: TPopupMenu;
```

#### Description

It is similar to TControl.PopupMenu property.

If there are no popup menu assigned to this property default popup menu is created. In this case you may use PopupMenuFilter to define possible item groups in default popup menu.

See TControl.PopupMenu for details.

### 1.15.1.4.1.58 TzFormDesigner.PopupMenuFilter Property

Specifies which categories of items may be used to form default popup menu.

```
property PopupMenuFilter: TLocalMenuFilters;
```

#### Description

### 1.15.1.4.1.59 TzFormDesigner.ReadOnly Property

Set Read Only mode.

```
property ReadOnly: Boolean;
```

#### Description

Set ReadOnly to True to disable any changes in designer. In read only mode there is no popup menu, all changes to controls are disabled (moving, resizing, etc.), component editors are disabled.

### 1.15.1.4.1.60 TzFormDesigner.SelMarker Property

Selection markers manager.

```
property SelMarker: TzBoundCtrl;
```

#### Description

Use this object to customize shape and colors of selection markers. Selection markers are visible when only one component is selected.

#### 1.15.1.4.1.61 TzFormDesigner.ShowCaptions Property

Specifies whether non-visual component icons captions are visible.

```
property ShowCaptions: Boolean;
```

##### Description

When ShowCaption is True non-visual components icons have captions with their names below them.

For data modules captions are always visible

#### 1.15.1.4.1.62 TzFormDesigner.ShowHints Property

Specifies showing of design hints.

```
property ShowHints: Boolean;
```

##### Description

Use this property to enable/disable designer's hints.

#### 1.15.1.4.1.63 TzFormDesigner.SnapToGrid Property

Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines.

```
property SnapToGrid: Boolean;
```

##### Description

#### 1.15.1.4.1.64 TzFormDesigner.StoreEvents Property

Specifies whether designer should process events storage.

```
property StoreEvents: Boolean;
```

##### Description

When StorageEvents is True events associations are save to Events (see page 189) property. These events are saved in resource (DFM file or stream).

Otherwise you need to use OnSetScriptProc and OnGetScriptProc events to process events associations manually.

#### 1.15.1.4.1.65 TzFormDesigner.TabOrderIcons Property

Properties of tab order icons. These controls are shown over children control when designer is in "Show Tab Order" mode.

```
property TabOrderIcons: TTabOrderIcons;
```

#### 1.15.1.4.1.66 TzFormDesigner.Target Property

Specifies object that is edited by the designer.

```
property Target: TComponent;
```

##### Description

Setting this property affects to Root, ContainerWindow and Form properties.

##### Possible Target types are following:

1. TCustomForm - form designing.

##### In this case:

Root = ContainerWindow = Form = Target.



2. TWinControl, owned by the form. This control and all its children controls are edited. Also non-visual components owned by the form may be edited too if AllowComponents is True.

**In this case:**

```
Root = Form = (Target.Owner as TCustomForm);
```

```
ContainerWindow = (Target as TWinControl);
```

3. TWinControl, not owned by the form, for example, TQuickReport.

This control and all owned by it components are edited.

In this case temporary form is created as container in design mode.

Client size of this form is equal to size of the Target.

Changing form size does not change size of the Target.

**In this case:**

```
Form = temporary internal Form;
```

```
Root = Target;
```

```
ContainerWindow = (Target as TWinControl);
```

4. TCustomFrame.

Identical to case 3, but changing form size changes size of the frame.

5. TDataModule.

Data module and all its components are edited.

For editing of data modules temporary Form and temporary container are created, in which designing are performed.

**In this case:**

```
Root = Target;
```

```
Form = temporary internal Form;
```

```
ContainerWindow = temporary internal Container;
```

#### 1.15.1.4.1.67 TzFormDesigner.TextEditMode Property

Sets in-place text editing mode of designer.

```
property TextEditMode: Boolean;
```

##### Description

Set TextEditMode to True to enable in-place editing of control's texts. In this mode labels, button captions, list box items, combo box items and many others text properties may be edited directly on the form. It will give more interactivity to design process.

To activate in-place editor user needs to click on text string on the control or press Enter, when this control is selected.

In-place editors are handled by special design classes derived from

1.15.1.4.1.68 TzFormDesigner.UndoLimit Property

Specifies the number of changes that can be undone.

```
property UndoLimit: integer;
```

Description






Use UndoLimit to restrict undo records list. If UndoLimit is 0, undo operation is disabled.

Default value is 16.


1.15.2 Structs, Records, Enums

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

Enumerations

Enumeration	Description
 TBufferizedType (see page 229)	Determines how object were placed into buffer
 TCompAlign (see page 229)	Specifies how selected will be aligned. This type is used for both vertical and horizontal alignment.
 TCompSize (see page 230)	Specifies type of selected component sizing.
 TGuidelinesStyle (see page 230)	Guidelines options.
 TLocalMenuFilter (see page 231)	Popup menu groups

Legend

	Enumeration
-------------------------------------------------------------------------------------	-------------

1.15.2.1 ed\_Designer.TBufferizedType Enumeration

Determines how object were placed into buffer

```
TBufferizedType = (  
    btCopy,  
    btCut  
);
```

File

ed\_Designer

Members

Members	Description
btCopy	objects were copied
btCut	objects were cutted

Description

- btCopy - objects were copied
- btCut - objects were cut

TBufferizedType is used to calculate smart offset when pasting components back to the form.

1.15.2.2 ed\_Designer.TCompAlign Enumeration

Specifies how selected will be aligned. This type is used for both vertical and horizontal alignment.

```
TCompAlign = (  
    ca_None,  
    caMin,
```

```
    caCenters,
    caMax,
    caSpaceEq,
    caCenterWnd
);
```

File

ed\_Designer

Members

Members	Description
ca_None	Does not change the alignment of the component
caMin	Lines up the left (top) edges of the selected components.
caCenters	Lines up the centers of the selected components
caMax	Lines up the right (bottom) edges of the selected components.
caSpaceEq	Lines up the selected components equidistant from each other.
caCenterWnd	Lines up the selected components with the center of the window

1.15.2.3 ed\_Designer.TCompSize Enumeration

Specifies type of selected component sizing.

```
TCompSize = (
    csNone,
    csShrink,
    csGrow,
    csValue
);
```

File

ed\_Designer

Members

Members	Description
csNone	Does not change the size of the components.
csShrink	Resizes the group of components to the height or width of the smallest selected component.
csGrow	Resizes the group of components to the height or width of the largest selected component.
csValue	Sets a custom width (height) for the selected components.

1.15.2.4 ed\_Designer.TGuidelinesStyle Enumeration

Guidelines options.

```
TGuidelinesStyle = (
    glLeft,
    glHCenter,
    glRight,
    glTop,
    glVCenter,
    glBottom,
    glMouse,
    glKeyboard,
    glMoving,
    glSizing,
    glStatic,
    glMultiple,
    glMultipleSel
);
```

File

ed\_Designer

**Members**

Members	Description
glLeft	Specifies whether guideline can be shown for left edges of controls.
glHCenter	Specifies whether guideline can be shown for horizontal centers of controls.
glRight	Specifies whether guideline can be shown for right edges of controls.
glTop	Specifies whether guideline can be shown for top edges of controls.
glVCenter	Specifies whether guideline can be shown for vertical centers of controls.
glBottom	Specifies whether guideline can be shown for bottom edges of controls.
glMouse	Specifies whether guidelines can be shown when moving or sizing by mouse.
glKeyboard	Specifies whether guidelines can be shown when Control or Shift keys are pressed, i.e. when moving or sizing by keyboard.
glMoving	Specifies whether guidelines can be shown when moving selected controls.
glSizing	Specifies whether guidelines can be shown when sizing selected control.
glStatic	Specifies whether guidelines can be shown for selected controls when no actions are perform on them, i.e. in static mode.
glMultiple	Specifies whether multiple guidelines can be shown, otherwise only single guideline will be displayed.
glMultipleSel	Specifies whether guidelines can be shown when more than one control is selected.

**1.15.2.5 ed\_Designer.TLocalMenuFilter Enumeration**

```
TLocalMenuFilter = (
    lmModule,
    lmComponent,
    lmDesigner
);
```

**File**

ed\_Designer

**Members**

Members	Description
lmModule	Include custom module menu items.
lmComponent	Menu items for working with components.
lmDesigner	Menu items for working with controls.

**Description**

Popup menu groups

**1.15.3 Types**

The following table lists types in this documentation.

**Types**

Type	Description
TComponentEvent ( <a href="#">see page 232</a> )	Event type for events on manipulations with components.
TCreateComponentEvent ( <a href="#">see page 232</a> )	See
TCreateFrameEvent ( <a href="#">see page 232</a> )	See TzCustomFormDesigner.OnCreateFrame Event ( <a href="#">see page 197</a> )
TCreateIconEvent ( <a href="#">see page 232</a> )	See TzCustomFormDesigner.OnCreateIcon Event ( <a href="#">see page 197</a> )
TCreateMethodEvent ( <a href="#">see page 233</a> )	See TzCustomFormDesigner.OnCreateMethod Event ( <a href="#">see page 198</a> )
TDrawControlEvent ( <a href="#">see page 233</a> )	See TzFormDesigner.OnDrawControl Event ( <a href="#">see page 220</a> ).
TGetComponentHintEvent ( <a href="#">see page 233</a> )	See TzCustomFormDesigner.OnGetComponentHint Event ( <a href="#">see page 198</a> )
TGetMethodNamesEvent ( <a href="#">see page 233</a> )	See TzCustomFormDesigner.OnGetMethodNames Event ( <a href="#">see page 198</a> )
TGetObjNameEvent ( <a href="#">see page 233</a> )	See TzCustomFormDesigner.OnGetObjectName Event ( <a href="#">see page 195</a> )
TGetScriptProcEvent ( <a href="#">see page 234</a> )	See TzCustomFormDesigner.OnGetScriptProc Event ( <a href="#">see page 199</a> )

TGuidelinesStyles (see page 234)	Set of TGuidelinesStyle (see page 230).
THandleActionEvent (see page 234)	See TzCustomFormDesigner.OnExecuteAction Event (see page 195)
TLocalMenuFilters (see page 234)	Set of TLocalMenuFilter (see page 231)
TNotificationEvent (see page 234)	See TzCustomFormDesigner.OnNotification Event (see page 199)
TRenameEvent (see page 235)	See TzCustomFormDesigner.OnRenameMethod Event (see page 200)
TRenameMethodEvent (see page 235)	See TzCustomFormDesigner.OnRenameMethod Event (see page 200)
TSetNameEvent (see page 235)	See TzCustomFormDesigner.OnSetNewName Event (see page 197)
TSetScriptProcEvent (see page 235)	See TzCustomFormDesigner.OnSetScriptProc Event (see page 200)
TShowMethodEvent (see page 235)	See TzCustomFormDesigner.OnShowMethod Event (see page 201)
TUndoRecEvent (see page 236)	See TzCustomFormDesigner.OnPopUndo Event (see page 196) and TzCustomFormDesigner.OnPushUndo Event (see page 197)
TValidateMethodEvent (see page 236)	See TzCustomFormDesigner.OnValidateMethod Event (see page 201)

### 1.15.3.1 ed\_Designer.TComponentEvent Type

```
TComponentEvent = procedure (Sender: TObject; Component: TComponent; var Accept: Boolean) of object;
```

#### File

ed\_Designer

#### Description

Event type for events on manipulations with components.

### 1.15.3.2 ed\_Designer.TCreateComponentEvent Type

```
TCreateComponentEvent = procedure (Sender: TObject; AClass: TComponentClass; AOwner: TComponent; var Instance: TComponent) of object;
```

#### File

ed\_Designer

#### Description

See

### 1.15.3.3 ed\_Designer.TCreateFrameEvent Type

```
TCreateFrameEvent = procedure (Sender: TObject; Root: TComponent; var Frame: TComponent) of object;
```

#### File

ed\_Designer

#### Description

See TzCustomFormDesigner.OnCreateFrame Event (see page 197)

### 1.15.3.4 ed\_Designer.TCreateIconEvent Type

```
TCreateIconEvent = procedure (Sender: TObject; Component: TComponent; var AllowCreate: Boolean) of object;
```

#### File

ed\_Designer

**Description**

See TzCustomFormDesigner.OnCreatelcon Event (🔗 see page 197)

### 1.15.3.5 ed\_Designer.TCreateMethodEvent Type

```
TCreateMethodEvent = procedure (Sender: TObject; const Name: string; TypeData: PTypeData;  
var Method: TMethod) of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnCreateMethod Event (🔗 see page 198)

### 1.15.3.6 ed\_Designer.TDrawControlEvents Type

See TzFormDesigner.OnDrawControl Event (🔗 see page 220).

```
TDrawControlEvents = procedure (Sender: TObject; Control: TControl; DC: HDC) of object;
```

**File**

ed\_Designer

### 1.15.3.7 ed\_Designer.TGetComponentHintEvent Type

```
TGetComponentHintEvent = procedure (Sender: TObject; Component: TComponent; var Hint:  
string; var ShowHint: Boolean) of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnGetComponentHint Event (🔗 see page 198)

### 1.15.3.8 ed\_Designer.TGetMethodNamesEvent Type

```
TGetMethodNamesEvent = procedure (Sender: TObject; TypeData: PTypeData; Proc: TGetStrProc)  
of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnGetMethodNames Event (🔗 see page 198)

### 1.15.3.9 ed\_Designer.TGetObjNameEvent Type

See TzCustomFormDesigner.OnGetObjectName Event (🔗 see page 195)

```
TGetObjNameEvent = procedure (Sender: TObject; Obj: TObject; var ObjName: string) of object;
```

**File**

ed\_Designer

### 1.15.3.10 ed\_Designer.TGetScriptProcEvent Type

```
TGetScriptProcEvent = procedure (Sender: TObject; Instance: TObject; pInfo: PPropInfo; var ProcName: string) of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnGetScriptProc Event (see page 199)

### 1.15.3.11 ed\_Designer.TGuidelinesStyles Type

Set of TGuidelinesStyle (see page 230).

```
TGuidelinesStyles = set of TGuidelinesStyle;
```

**File**

ed\_Designer

### 1.15.3.12 ed\_Designer.THandleActionEvent Type

See TzCustomFormDesigner.OnExecuteAction Event (see page 195)

```
THandleActionEvent = procedure (Sender: TObject; Action: TBasicAction; var Handled: Boolean) of object;
```

**File**

ed\_Designer

### 1.15.3.13 ed\_Designer.TLocalMenuFilters Type

```
TLocalMenuFilters = set of TLocalMenuFilter;
```

**File**

ed\_Designer

**Description**

Set of TLocalMenuFilter (see page 231)

### 1.15.3.14 ed\_Designer.TNotificationEvent Type

```
TNotificationEvent = procedure (Sender: TObject; AnObject: TPersistent; Operation: TOperation) of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnNotification Event (see page 199)

### 1.15.3.15 ed\_Designer.TRenameEvent Type

```
TRenameEvent = procedure (Sender: TObject; Component: TComponent; const NewName: string;  
var Accept: Boolean) of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnRenameMethod Event (🔗 see page 200)

### 1.15.3.16 ed\_Designer.TRenameMethodEvent Type

```
TRenameMethodEvent = procedure (Sender: TObject; const CurName, NewName: string) of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnRenameMethod Event (🔗 see page 200)

### 1.15.3.17 ed\_Designer.TSetNameEvent Type

See TzCustomFormDesigner.OnSetNewName Event (🔗 see page 197)

```
TSetNameEvent = procedure (Sender: TObject; Component: TComponent; var Name: string) of object;
```

**File**

ed\_Designer

### 1.15.3.18 ed\_Designer.TSetScriptProcEvent Type

```
TSetScriptProcEvent = procedure (Sender: TObject; Instance: TObject; pInfo: PPropInfo;  
const EventProc: string) of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnSetScriptProc Event (🔗 see page 200)

### 1.15.3.19 ed\_Designer.TShowMethodEvent Type

```
TShowMethodEvent = procedure (Sender: TObject; const MethodName: string) of object;
```

**File**

ed\_Designer

**Description**

See TzCustomFormDesigner.OnShowMethod Event (🔗 see page 201)



### 1.15.3.20 ed\_Designer.TUndoRecEvent Type

See TzCustomFormDesigner.OnPopUndo Event (see page 196) and TzCustomFormDesigner.OnPushUndo Event (see page 197)

```
TUndoRecEvent = procedure (Sender: TObject; Stream: TStream) of object;
```

**File**  
ed\_Designer

### 1.15.3.21 ed\_Designer.TValidateMethodEvent Type

```
TValidateMethodEvent = procedure (Sender: TObject; TypeData: PTypeData; ARoot: TObject; MethAddr: pointer; const MethodName: string; var Accept: Boolean) of object;
```

**File**  
ed\_Designer

**Description**  
See TzCustomFormDesigner.OnValidateMethod Event (see page 201)

## 1.15.4 Constants

The following table lists constants in this documentation.

**Constants**

Constant	Description
DM_POSCHANGED (see page 236)	The DM_POSCHANGED message is sent to a window whose size, position, or place in the Z order has changed to update selection markers around selected components.
sLineBreak (see page 236)	Used for Delphi5

### 1.15.4.1 ed\_Designer.DM\_POSCHANGED Constant

The DM\_POSCHANGED message is sent to a window whose size, position, or place in the Z order has changed to update selection markers around selected components.

```
DM_POSCHANGED = $C002;
```

**File**  
ed\_Designer

**Description**

### 1.15.4.2 ed\_Designer.sLineBreak Constant

Used for Delphi5  
sLineBreak = #13#10;

**File**  
ed\_Designer

**Description**

## 1.16 ed\_dsncont Namespace

### 1.16.1 Classes

The following table lists classes in this documentation.

**Classes**

Class	Description
TDesignSurface (🔗 see page 237)	Design surface window.

#### 1.16.1.1 TDesignSurface Class

Design surface window.

**Class Hierarchy**

```
TDesignSurface = class(TCustomControl);
```

**File**

ed\_dsncont

**Description**

Serves as deigned form container. Blocks activation of the form.Redirects keyboard input to designer.







Assign design surface to TzCustomFormDesigner.DesignSurface Property (🔗 see page 188). Designer automatically puts designed form on the surface.

**Members****TDesignSurface Methods**





TDesignSurface Methods	Description
🔗 Activate (🔗 see page 238)	Activates associated designer.
🔗 AdjustScroll (🔗 see page 238)	Adjusts scrollbars and form position in the container.
🔗 DoSizing (🔗 see page 238)	Called by designer when form was resized.
🔗 ExecuteAction (🔗 see page 238)	Invokes an action with the component as its target.
🔗 UpdateAction (🔗 see page 239)	Updates an action component to reflect the current state of the component.

**TDesignSurface Properties**





TDesignSurface Properties	Description
🔗 DsnShowFrame (🔗 see page 239)	Designer sets this property to show/hide frame around form.
🔗 FlatScrollBars (🔗 see page 239)	Specifies type of used scroll bars.
🔗 Form (🔗 see page 240)	Specifies attached form.
🔗 FormOrigin (🔗 see page 240)	Specifies offset of the form relative to client origin of design surface.
🔗 FrameSize (🔗 see page 240)	Specifies width of the frame line.
🔗 HideFormBorders (🔗 see page 240)	Specifies whether form caption and border should be hidden.
🔗 RulerClientArea (🔗 see page 240)	Specifies whether ruler displays scale only for client area.
🔗 ScrollPos (🔗 see page 240)	Specifies current scrolling position.

 ShowFrame (  ) see page 240)	Specifies whether frame around form may be visible.
 ShowRuler (  ) see page 241)	Specifies whether rulers are visible.
 UseUnits (  ) see page 241)	Specifies units used by rulers.























## Legend

	Method
	virtual
	Property
	read only

## TDesignSurface Methods

TDesignSurface Methods	Description
 Activate (  ) see page 238)	Activates associated designer.
 AdjustScroll (  ) see page 238)	Adjusts scrollbars and form position in the container.
 DoSizing (  ) see page 238)	Called by designer when form was resized.
 ExecuteAction (  ) see page 238)	Invokes an action with the component as its target.
 UpdateAction (  ) see page 239)	Updates an action component to reflect the current state of the component.

## TDesignSurface Properties

TDesignSurface Properties	Description
 DsnShowFrame (  ) see page 239)	Designer sets this property to show/hide frame around form.
 FlatScrollBars (  ) see page 239)	Specifies type of used scroll bars.
 Form (  ) see page 240)	Specifies attached form.
 FormOrigin (  ) see page 240)	Specifies offset of the form relative to client origin of design surface.
 FrameSize (  ) see page 240)	Specifies width of the frame line.
 HideFormBorders (  ) see page 240)	Specifies whether form caption and border should be hidden.
 RulerClientArea (  ) see page 240)	Specifies whether ruler displays scale only for client area.
 ScrollPos (  ) see page 240)	Specifies current scrolling position.
 ShowFrame (  ) see page 240)	Specifies whether frame around form may be visible.
 ShowRuler (  ) see page 241)	Specifies whether rulers are visible.
 UseUnits (  ) see page 241)	Specifies units used by rulers.

## 1.16.1.1.1 TDesignSurface Methods

### 1.16.1.1.1.1 TDesignSurface.Activate Method

Activates associated designer.

```
procedure Activate;
```

### 1.16.1.1.1.2 TDesignSurface.AdjustScroll Method

Adjusts scrollbars and form position in the container.

```
procedure AdjustScroll;
```

### 1.16.1.1.1.3 TDesignSurface.DoSizing Method

Called by designer when form was resized.

```
procedure DoSizing(var Rect: TRect);
```

## Description

### 1.16.1.1.1.4 TDesignSurface.ExecuteAction Method

Invokes an action with the component as its target.

```
function ExecuteAction(Action: TBasicAction): Boolean; override;
```

## Description

When the user invokes an action, VCL makes a series of calls to respond to that action. First, it generates an OnExecute event of the action list that contains the action. If the action list does not handle the OnExecute event, then the action is routed to the Application object's ExecuteAction method, which invokes the OnActionExecute event handler. If the OnActionExecute event handler does not handle the action, then it is routed to the action's OnExecute event handler. If that does not handle the action, the active control's ExecuteAction method is called.

The Action parameter specifies the action that was invoked. ExecuteAction returns true if the action was successfully dispatched, and false if the component could not handle the action. If ExecuteAction returns false for the active control, VCL calls the active form's ExecuteAction method. If this returns false, VCL tries all active controls in the form. If these all return false, VCL repeats the process with the main form, if that is different from the active form.

### 1.16.1.1.1.5 TDesignSurface.UpdateAction Method

Updates an action component to reflect the current state of the component.

```
function UpdateAction(Action: TBasicAction): Boolean; override;
```

## Description

When the application is idle, VCL makes a series of calls to update the properties (such as whether it is enabled, checked, and so on) of every action that is linked to a visible control or menu item. First, VCL generates an OnUpdate event of the action list that contains the action. If the action list does not handle the OnUpdate event, then the action is routed to the Application object's UpdateAction method, which invokes the OnActionUpdate event handler. If the OnActionUpdate event handler does not update the action, then it is routed to the action's OnUpdate event handler. If that does not update the action, the active control's UpdateAction method is called.

The Action parameter specifies the action component that should be updated. UpdateAction returns true if the action component now reflects the state of the component, and false if it did not know how to update the action. If UpdateAction returns false for the active component, VCL calls the active form's

s UpdateAction method.

Do not call UpdateAction. It is called automatically when the application is idle. As implemented in TComponent, UpdateAction allows the action to update itself with the component as a target. Descendants can override this method to perform updates that reflect class-specific properties or states.

### 1.16.1.1.2 TDesignSurface Properties

#### 1.16.1.1.2.1 TDesignSurface.DsnShowFrame Property

Designer sets this property to show/hide frame around form.

```
property DsnShowFrame: Boolean;
```

## Description

Frame around form is displayed when ShowFrame (☐ see page 240) = True and DsnShowFrame = True.

Designer shows frame when there are no selected controls, i.e. when entire form is selected.

#### 1.16.1.1.2.2 TDesignSurface.FlatScrollBars Property

Specifies type of used scroll bars.

```
property FlatScrollBars: Boolean;
```

**Description**

If FlatScrollBars is True flat scroll bar controls are used, otherwise standard windows scroll bars are used.

**1.16.1.1.2.3 TDesignSurface.Form Property**

Specifies attached form.

```
property Form: TCustomForm;
```

**Description****1.16.1.1.2.4 TDesignSurface.FormOrigin Property**

Specifies offset of the form relative to client origin of design surface.

```
property FormOrigin: TPoint;
```

**Description****1.16.1.1.2.5 TDesignSurface.FrameSize Property**

Specifies width of the frame line.

```
property FrameSize: integer;
```

**Description****1.16.1.1.2.6 TDesignSurface.HideFormBorders Property**

Specifies whether form caption and border should be hidden.

```
property HideFormBorders: Boolean;
```

**Description**

This property allows hiding of form border and caption during design. It is useful when you are implementing flat pane designer.

**1.16.1.1.2.7 TDesignSurface.RulerClientArea Property**

Specifies whether ruler displays scale only for client area.

```
property RulerClientArea: Boolean;
```

**1.16.1.1.2.8 TDesignSurface.ScrollPos Property**

Specifies current scrolling position.

```
property ScrollPos: TPoint;
```

**Description****1.16.1.1.2.9 TDesignSurface.ShowFrame Property**

Specifies whether frame around form may be visible.

```
property ShowFrame: Boolean;
```

Description

When this property is True, designer can show frame by setting DsnShowFrame (see page 239) property to True. Form origin is increased on FrameSize (see page 240) pixels.

1.16.1.1.2.10 TDesignSurface.ShowRuler Property

Specifies whether rulers are visible.

**property** ShowRuler: Boolean;

Description

Use this property to hide/show rulers.

1.16.1.1.2.11 TDesignSurface.UseUnits Property

Specifies units used by rulers.


**property** UseUnits: TRulerUnits;

Description

1.16.2 Structs, Records, Enums

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

Enumerations

Enumeration	Description
 TRulerUnits (see page 241)	Units for design surface ruler.

Legend

	Enumeration
-------------------------------------------------------------------------------------	-------------

1.16.2.1 ed\_dsncont.TRulerUnits Enumeration

```
TRulerUnits = (  
    ruCentimeters,  
    ruPixels,  
    ruInches  
);
```

File

ed\_dsncont

Members

Members	Description
ruCentimeters	Centimeters
ruPixels	Pixels
ruInches	Inches

Description

Units for design surface ruler.

# 1.17 ed\_RegComps Namespace

## 1.17.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
Frames (🔗 see page 242)	Special class to support work with frames in design-mode
TComponentClassInfo (🔗 see page 242)	Class description for registered component.
TCustomModuleInfo (🔗 see page 245)	Description of the custom modules classes.
TFrameInfo (🔗 see page 245)	Frame's registration information
TPackageInfo (🔗 see page 246)	Package description for registered package.
TPackageMng (🔗 see page 248)	TPackageMng is a global registration object for components, property and component editors, actions and so on.

### 1.17.1.1 Frames Class

Special class to support work with frames in design-mode

Class Hierarchy



```
Frames = class(TComponent);
```

File

ed\_RegComps

Description

This class is used internally to provide working with instances of TFrame descendants.

### 1.17.1.2 TComponentClassInfo Class

Class description for registered component.

Class Hierarchy



```
TComponentClassInfo = class;
```

File

ed\_RegComps

Description











TComponentClassInfo encapsulates information on registered component.

Instances of this class is being created for every component in the registered package while RegisterComponents procedure is called by the application or in Register method in the packages.





**Members****TComponentClassInfo Methods**

TComponentClassInfo Methods	Description
 Create ( <a href="#">see page 243</a> )	Creates instance of this class.
 Destroy ( <a href="#">see page 243</a> )	Destroys instance of this class.
 IsIconValid ( <a href="#">see page 244</a> )	Returns true if component icon is valid image.

**TComponentClassInfo Properties**

TComponentClassInfo Properties	Description
  ComponentClass ( <a href="#">see page 244</a> )	Reference to the component class.
 DisplayName ( <a href="#">see page 244</a> )	Display name of component class.
 Hidden ( <a href="#">see page 244</a> )	Specifies whether component class is visible on the component palette.
 Icon ( <a href="#">see page 244</a> )	Specifies bitmap image that is displayed in the component palette.
  InitPage ( <a href="#">see page 244</a> )	Specifies the initial palette page name (specified during registration).
  Module ( <a href="#">see page 244</a> )	Reference to package description where this component class has been registered from.
 Page ( <a href="#">see page 245</a> )	Palette page name where component's icon has being located.











**Legend**

	Constructor
	virtual
	Property
	read only

**TComponentClassInfo Methods**

TComponentClassInfo Methods	Description
 Create ( <a href="#">see page 243</a> )	Creates instance of this class.
 Destroy ( <a href="#">see page 243</a> )	Destroys instance of this class.
 IsIconValid ( <a href="#">see page 244</a> )	Returns true if component icon is valid image.

**TComponentClassInfo Properties**

TComponentClassInfo Properties	Description
  ComponentClass ( <a href="#">see page 244</a> )	Reference to the component class.
 DisplayName ( <a href="#">see page 244</a> )	Display name of component class.
 Hidden ( <a href="#">see page 244</a> )	Specifies whether component class is visible on the component palette.
 Icon ( <a href="#">see page 244</a> )	Specifies bitmap image that is displayed in the component palette.
  InitPage ( <a href="#">see page 244</a> )	Specifies the initial palette page name (specified during registration).
  Module ( <a href="#">see page 244</a> )	Reference to package description where this component class has been registered from.
 Page ( <a href="#">see page 245</a> )	Palette page name where component's icon has being located.

**1.17.1.2.1 TComponentClassInfo Methods****1.17.1.2.1.1 TComponentClassInfo.Create Constructor**

Creates instance of this class.

```
constructor Create(AClass: TComponentClass; const APage: string);
```

**1.17.1.2.1.2 TComponentClassInfo.Destroy Destructor**

Destroys instance of this class.

```
destructor Destroy; override;
```

**Description**



### 1.17.1.2.1.3 TComponentClassInfo.IsIconValid Method

Returns true if component icon is valid image.

```
function IsIconValid: Boolean;
```

### 1.17.1.2.2 TComponentClassInfo Properties

#### 1.17.1.2.2.1 TComponentClassInfo.ComponentClass Property

Reference to the component class.

```
property ComponentClass: TComponentClass;
```

##### Description

ComponentClass is a reference to component class whose information TComponentClassInfo (see page 242) instance keeps.

#### 1.17.1.2.2.2 TComponentClassInfo.DisplayName Property

Display name of component class.

```
property DisplayName: WideString;
```

##### Description

Display name is used in component palette and palette tool list. It is initialized with the component class name, but it may be changed at runtime.

#### 1.17.1.2.2.3 TComponentClassInfo.Hidden Property

Specifies whether component class is visible on the component palette.

```
property Hidden: Boolean;
```

##### Description

Use this property to get or set visibility of the component icon on the palette.

#### 1.17.1.2.2.4 TComponentClassInfo.Icon Property

Specifies bitmap image that is displayed in the component palette.

```
property Icon: TBitmap;
```

##### Description

Use this property to get or set component's icon on the palette.

#### 1.17.1.2.2.5 TComponentClassInfo.InitPage Property

Specifies the initial palette page name (specified during registration).

```
property InitPage: string;
```

##### Description

This field means Palette page name by default and is specified during registration process. Real name may differ because of palette customizing process.

#### 1.17.1.2.2.6 TComponentClassInfo.Module Property

Reference to package description where this component class has been registered from.

```
property Module: TPackageInfo;
```

Description

Use this field to access to module's info where this component from. This field may be nil if component has no owner's module, i.e. 'Frames (see page 242)'.

1.17.1.2.2.7 TComponentClassInfo.Page Property

Palette page name where component's icon has being located.

```
property Page: string;
```

Description

This page may differ from InitPage (see page 244), because user may customize component palette.

1.17.1.3 TCustomModuleInfo Class

Description of the custom modules classes.

Class Hierarchy



```
TCustomModuleInfo = class;
```

File

ed\_RegComps

Description

Custom modules is registered to support custom design environment. For example, 'Quick report' library has own custom module.

1.17.1.4 TFrameInfo Class

Frame's registration information

Class Hierarchy



```
TFrameInfo = class;
```

File

ed\_RegComps

Members

TFrameInfo Properties

TFrameInfo Properties	Description
FrameClass (see page 246)	Frame class, instance of which is created when frame inserted on the form.
FrameResource (see page 246)	Frame resource which is used to initialize new frame instance.

Legend

	Property
	read only

TFrameInfo Properties

TFrameInfo Properties	Description
FrameClass (see page 246)	Frame class, instance of which is created when frame inserted on the form.
FrameResource (see page 246)	Frame resource which is used to initialize new frame instance.

## 1.17.1.4.1 TFrameInfo Properties

### 1.17.1.4.1.1 TFrameInfo.FrameClass Property

Frame class, instance of which is created when frame inserted on the form.

**property** FrameClass: TCustomFrameClass;

### 1.17.1.4.1.2 TFrameInfo.FrameResource Property

Frame resource which is used to initialize new frame instance.

**property** FrameResource: TCustomFrame;

## 1.17.1.5 TPackageInfo Class

Package description for registered package.

### Class Hierarchy

ed\_RegComps.TPackageInfo

TPackageInfo = **class**;

### File

ed\_RegComps

### Description













Instances of this class created during loading packages using PackageMng.AddPackage.

### Members





#### TPackageInfo Methods

TPackageInfo Methods	Description
 Create (  see page 247)	Creates and initializes instance of TPackageInfo class.
 Destroy (  see page 247)	Destroys instance of TPackageInfo and releases all of resources.

#### TPackageInfo Properties

TPackageInfo Properties	Description
 Active (  see page 247)	Specifies whether component of this package available in component palette.
 Description (  see page 247)	Specifies description string from the package header.
 FileName (  see page 247)	Specifies file name of the package.
 Handle (  see page 247)	Specifies package handle.
 Requires (  see page 247)	Specifies list of packages those are loaded with this package.
 Units (  see page 248)	Specifies list of units in the package.

### Legend






	Constructor
	virtual
	Property
	read only

#### TPackageInfo Methods

TPackageInfo Methods	Description
 Create (  see page 247)	Creates and initializes instance of TPackageInfo class.
 Destroy (  see page 247)	Destroys instance of TPackageInfo and releases all of resources.

#### TPackageInfo Properties

TPackageInfo Properties	Description
 Active (  see page 247)	Specifies whether component of this package available in component palette.

 <b>Description</b> (see page 247)	Specifies description string from the package header.
 <b>FileName</b> (see page 247)	Specifies file name of the package.
 <b>Handle</b> (see page 247)	Specifies package handle.
 <b>Requires</b> (see page 247)	Specifies list of packages those are loaded with this package.
 <b>Units</b> (see page 248)	Specifies list of units in the package.

## 1.17.1.5.1 TPackageInfo Methods

### 1.17.1.5.1.1 TPackageInfo.Create Constructor

Creates and initializes instance of TPackageInfo (see page 246) class.

```
constructor Create(AFileName: string);
```

### 1.17.1.5.1.2 TPackageInfo.Destroy Destructor

Destroys instance of TPackageInfo (see page 246) and releases all of resources.

```
destructor Destroy; override;
```

## 1.17.1.5.2 TPackageInfo Properties

### 1.17.1.5.2.1 TPackageInfo.Active Property

Specifies whether component of this package available in component palette.

```
property Active: Boolean;
```

#### Description

Set Active to True to let components of this package be available on component palette and vice versa.

### 1.17.1.5.2.2 TPackageInfo.Description Property

Specifies description string from the package header.

```
property Description: string;
```

#### Description

This is read-only property.

### 1.17.1.5.2.3 TPackageInfo.FileName Property

Specifies file name of the package.

```
property FileName: string;
```

#### Description

This is read-only property.

### 1.17.1.5.2.4 TPackageInfo.Handle Property

Specifies package handle.

```
property Handle: HMODULE;
```

#### Description

This is read-only property.

### 1.17.1.5.2.5 TPackageInfo.Requires Property

Specifies list of packages those are loaded with this package.

**property Requires:** TStrings;

### Description

This is read-only property.

## 1.17.1.5.2.6 TPackageInfo.Units Property

Specifies list of units in the package.

**property Units:** TStrings;

### Description

This is read-only property.

## 1.17.1.6 TPackageMng Class

TPackageMng is a global registration object for components, property and component editors, actions and so on.

### Class Hierarchy

ed\_RegComps.TPackageMng

TPackageMng = **class**;

### File

ed\_RegComps

### Description

TPackageMng is used to control and registration designer objects such as: components, property and component editors, actions ...

This class performs loading packages, registration packages (calls Register method of each unit in the package), reading/saving its state from the registry, registration of frames, customizing palette and packages list.







### Notes

Do not create (see page 250) objects of this class. Use global variable PackageMng (see page 258) to access registration information.








### Members

#### TPackageMng Methods




TPackageMng Methods	Description
◆ AddComponent (see page 250)	Adds new component information.
◆ AddPackage (see page 250)	Loads package (usually *.bpl file).
◆ BeginUpdate (see page 250)	Suspends visual palette updating.
◆ Create (see page 250)	Creates and initializes global PackageMng (see page 258) object.
◆ CreateFrame (see page 251)	Display dialog to add new frame to the application.
◆ CustomizePackages (see page 251)	Display dialog to customize loaded package list.
◆ CustomizePalette (see page 251)	Display dialog to customize component palette.
◆ DeleteComponent (see page 251)	Deletes (unregisters) component information from manager.
◆ Destroy (see page 251)	Destroys global PackageMng (see page 258) object.
◆ EndUpdate (see page 252)	Re-enables visual palette updating.
◆ FindClass (see page 252)	Looks for registration information for the specified component class.
◆ FindClassName (see page 252)	Looks for registration information for the specified component class name.
◆ FindClassNameIdx (see page 252)	Looks for registration information for the specified component class name.
◆ FindPackage (see page 252)	Looks for registration information for the specified package name.
◆ GetCustomModule (see page 252)	Returns Custom Module object for specified Root.
◆ IsNolcon (see page 253)	Checks whether components of this class should not be displayed.
◆ LoadPaletteFromIni (see page 253)	Loads component palette settings from INI file.

 ReadRegInfo (see page 253)	Reads information about installed packages and component palette from the registry. Returns True if reading was successful.
 RegisterFrame (see page 253)	Registers frame class.
 RemoveEmptyPages (see page 253)	Removes all empty pages from component palette.
 RemovePackage (see page 254)	Removes all registration information associated with the package.
 RenamePage (see page 254)	Renames component panel page.
 ResetPalette (see page 254)	Resets component palette to its initial configuration.
 SavePaletteToIni (see page 254)	Saves component palette settings to INI file.
 SaveRegInfo (see page 254)	Saves information about installed packages and component palette in the registry. Returns True if saving was successful.
 SetComponentOrder (see page 255)	Moves component information to specified position in palette.






## TPackageMng Properties

TPackageMng Properties	Description
 AutoSave (see page 255)	Specifies whether packages information should be saved automatically.
 ComponentCount (see page 255)	Specifies number of registered component classes.
 Components (see page 255)	Specifies indexed access to registered components.
 FrameInfos (see page 255)	Specifies description list of registered component classes.
 Packages (see page 255)	Specifies list of loaded packages.
 Pages (see page 256)	Specifies list of palette pages.
 RegSubkey (see page 256)	Specifies registry key to which packages information will be saved\loaded.




## TPackageMng Events

TPackageMng Events	Description
 OnRegisterComponent (see page 256)	Occurs before component will be registered.
 OnRegisterComponentInfo (see page 256)	Occurs when new component class is registered in component palette.
 OnUnRegisterComponentInfo (see page 256)	Occurs when component information is unregistered, i.e. removed from component palette.
















## Legend

	Method
	virtual
	Property
	read only
	Event

## TPackageMng Events

TPackageMng Events	Description
 OnRegisterComponent (see page 256)	Occurs before component will be registered.
 OnRegisterComponentInfo (see page 256)	Occurs when new component class is registered in component palette.
 OnUnRegisterComponentInfo (see page 256)	Occurs when component information is unregistered, i.e. removed from component palette.

## TPackageMng Methods

TPackageMng Methods	Description
 AddComponent (see page 250)	Adds new component information.
 AddPackage (see page 250)	Loads package (usually *.bpl file).
 BeginUpdate (see page 250)	Suspends visual palette updating.
 Create (see page 250)	Creates and initializes global PackageMng (see page 258) object.
 CreateFrame (see page 251)	Display dialog to add new frame to the application.
 CustomizePackages (see page 251)	Display dialog to customize loaded package list.
 CustomizePalette (see page 251)	Display dialog to customize component palette.
 DeleteComponent (see page 251)	Deletes (unregisters) component information from manager.
 Destroy (see page 251)	Destroys global PackageMng (see page 258) object.
 EndUpdate (see page 252)	Re-enables visual palette updating.
 FindClass (see page 252)	Looks for registration information for the specified component class.
 FindClassName (see page 252)	Looks for registration information for the specified component class name.
 FindClassNameldx (see page 252)	Looks for registration information for the specified component class name.
 FindPackage (see page 252)	Looks for registration information for the specified package name.
 GetCustomModule (see page 252)	Returns Custom Module object for specified Root.

◆ IsNolcon (🔗 see page 253)	Checks whether components of this class should not be displayed.
◆ LoadPaletteFromIni (🔗 see page 253)	Loads component palette settings from INI file.
◆ ReadRegInfo (🔗 see page 253)	Reads information about installed packages and component palette from the registry. Returns True if reading was successful.
◆ RegisterFrame (🔗 see page 253)	Registers frame class.
◆ RemoveEmptyPages (🔗 see page 253)	Removes all empty pages from component palette.
◆ RemovePackage (🔗 see page 254)	Removes all registration information associated with the package.
◆ RenamePage (🔗 see page 254)	Renames component panel page.
◆ ResetPalette (🔗 see page 254)	Resets component palette to its initial configuration.
◆ SavePaletteToIni (🔗 see page 254)	Saves component palette settings to INI file.
◆ SaveRegInfo (🔗 see page 254)	Saves information about installed packages and component palette in the registry. Returns True if saving was successful.
◆ SetComponentOrder (🔗 see page 255)	Moves component information to specified position in palette.

## TPackageMng Properties

TPackageMng Properties	Description
📁 AutoSave (🔗 see page 255)	Specifies whether packages information should be saved automatically.
📁 ComponentCount (🔗 see page 255)	Specifies number of registered component classes.
📁 Components (🔗 see page 255)	Specifies indexed access to registered components.
📁 FrameInfos (🔗 see page 255)	Specifies description list of registered component classes.
📁 Packages (🔗 see page 255)	Specifies list of loaded packages.
📁 Pages (🔗 see page 256)	Specifies list of palette pages.
📁 RegSubkey (🔗 see page 256)	Specifies registry key to which packages information will be saved\loaded.

## 1.17.1.6.1 TPackageMng Methods

### 1.17.1.6.1.1 TPackageMng.AddComponent Method

Adds new component information.

```
procedure AddComponent(Cmp: TComponentClassInfo);
```

### 1.17.1.6.1.2 TPackageMng.AddPackage Method

Loads package (usually \*.bpl file).

```
function AddPackage(const Name: string): TPackageInfo;
```

#### Description

AddPackage loads package, calls procedures Register and RuntimeRegister for each unit in a package. Thus, all design-time objects are registered.

Function returns reference to the TPackageInfo (🔗 see page 246) object will be added.

### 1.17.1.6.1.3 TPackageMng.BeginUpdate Method

Suspends visual palette updating.

```
procedure BeginUpdate;
```

### 1.17.1.6.1.4 TPackageMng.Create Constructor

Creates and initializes global PackageMng (🔗 see page 258) object.

```
constructor Create;
```

#### Notes

Do not destroy (🔗 see page 251) this object! It will be destroyed automatically when program terminates.

### 1.17.1.6.1.5 TPackageMng.CreateFrame Method

Display dialog to add new frame to the application.

```
function CreateFrame(AOwner: TComponent): TCustomFrame;
```

#### Description

Displays "Frames (see page 242)" dialog, then creates instance of frame selected by user. Called by the designer when user is going to insert new frame.

### 1.17.1.6.1.6 TPackageMng.CustomizePackages Method

Display dialog to customize loaded package list.

```
function CustomizePackages: Boolean;
```

#### Description

Displays "Packages (see page 255)" dialog. In this dialog user may install or uninstall any package.

### 1.17.1.6.1.7 TPackageMng.CustomizePalette Method

Display dialog to customize component palette.

```
function CustomizePalette: Boolean;
```


#### Description

Displays "Customize palette" dialog. In this dialog user may customize palette pages (modify, create (see page 250), delete), hide/show any available component.

### 1.17.1.6.1.8 DeleteComponent Method

Deletes (unregisters) component information from manager.

#### Legend

	Method
-------------------------------------------------------------------------------------	--------

#### 1.17.1.6.1.8.1 TPackageMng.DeleteComponent Method (TComponentClass)

Deletes (unregisters) component information from manager by component class reference.

```
procedure DeleteComponent(Cls: TComponentClass); overload;
```

#### 1.17.1.6.1.8.2 TPackageMng.DeleteComponent Method (TComponentClassInfo)

Deletes (unregisters) component information from manager by reference.

```
procedure DeleteComponent(Cmp: TComponentClassInfo); overload;
```

#### 1.17.1.6.1.8.3 TPackageMng.DeleteComponent Method (integer)

Deletes (unregisters) component information from manager by index.

```
procedure DeleteComponent(Index: integer); overload;
```

### 1.17.1.6.1.9 TPackageMng.Destroy Destructor

Destroys global PackageMng (see page 258) object.

```
destructor Destroy; override;
```

#### Notes

Do not destroy this object! It will be destroyed automatically when program terminates.



### 1.17.1.6.1.10 TPackageMng.EndUpdate Method

Re-enables visual palette updating.

```
procedure EndUpdate;
```

### 1.17.1.6.1.11 TPackageMng.FindClass Method

Looks for registration information for the specified component class.

```
function FindClass(AClass: TClass): TComponentClassInfo;
```

#### Description

FindClass looks for TComponentClassInfo (see page 242) information for specified class. If no such information has found it returns **nil**.

### 1.17.1.6.1.12 TPackageMng.FindClassName Method

Looks for registration information for the specified component class name.

```
function FindClassName(AClassName: string): TComponentClassInfo;
```

#### Description

FindClassName is similar to FindClass (see page 252) method but look for TComponentClassInfo (see page 242) info via class type name.

If no such information has found it returns **nil**.

### 1.17.1.6.1.13 TPackageMng.FindClassNameIdx Method

Looks for registration information for the specified component class name.

```
function FindClassNameIdx(AClassName: string): integer;
```

#### Description

FindClassName (see page 252) is similar to FindClass (see page 252) method but looks for index of TComponentClassInfo (see page 242) info via class type name.

If no such information has found it returns **nil**.

### 1.17.1.6.1.14 TPackageMng.FindPackage Method

Looks for registration information for the specified package name.

```
function FindPackage(AFileName: string): TPackageInfo;
```

#### Description

Use FindPackage to find registration information for the specified package name.

If no such information has found it returns **nil**.

### 1.17.1.6.1.15 TPackageMng.GetCustomModule Method

Returns Custom Module object for specified Root.

```
function GetCustomModule(Root: TComponent; Designer: IDesigner): ICustomModule;
```

**Description****1.17.1.6.1.16 TPackageMng.IsNoIcon Method**

Checks whether components of this class should not be displayed.

```
function IsNoIcon(AClass: TClass): Boolean;
```

**Description**

For example, TMenuItem objects are not visible in designer. Such classes are registered using RegisterNoIcon function.

**1.17.1.6.1.17 LoadPaletteFromIni Method****1.17.1.6.1.17.1 TPackageMng.LoadPaletteFromIni Method (TCustomIniFile)**

Loads component palette settings from INI file.

```
procedure LoadPaletteFromIni(Ini: TCustomIniFile); overload;
```

**1.17.1.6.1.17.2 TPackageMng.LoadPaletteFromIni Method (string)**

Loads component palette settings from INI file.

```
procedure LoadPaletteFromIni(const FileName: string); overload;
```

**Description****1.17.1.6.1.18 TPackageMng.ReadRegInfo Method**

Reads information about installed packages and component palette from the registry. Returns True if reading was successful.

```
function ReadRegInfo(const RegKey: string = ''): Boolean;
```

**Description**

RegKey is a registry key.

RootKey is always HKEY\_LOCAL\_MACHINE.

**1.17.1.6.1.19 TPackageMng.RegisterFrame Method**

Registers frame class.

```
procedure RegisterFrame(AClass: TCustomFrameClass; AResource: TCustomFrame);
```

**Description**

All newly created frames in the designer will be initialized using AResource object.

The AClass parameter is a frame class is being registered.

The AResource parameter is a reference to object from which this frames is initialized.

**1.17.1.6.1.20 TPackageMng.RemoveEmptyPages Method**

Removes all empty pages from component palette.

```
procedure RemoveEmptyPages(All: Boolean = False);
```

**Description**

Removes all empty pages (without components) so component palette does not be jammed with useless elements.

**1.17.1.6.1.21 TPackageMng.RemovePackage Method**

Removes all registration information associated with the package.

```
procedure RemovePackage(pack: TPackageInfo);
```

**Description**

Package is not removed from memory. If you sure that this package is not use you may unload package using WIN API FreeLibrary.

**1.17.1.6.1.22 TPackageMng.RenamePage Method**

Renames component panel page.

```
procedure RenamePage(const OldName: string; const NewName: string);
```

**Description**

Use this method instead of changing Pages (see page 256) property. Each component class descriptor has Page property, renaming page updates Pages (see page 256) property and changes Page property in component class descriptors.

**1.17.1.6.1.23 TPackageMng.ResetPalette Method**

Resets component palette to its initial configuration.

```
procedure ResetPalette;
```

**1.17.1.6.1.24 SavePaletteToIni Method****1.17.1.6.1.24.1 TPackageMng.SavePaletteToIni Method (TCustomIniFile)**

Saves component palette settings to INI file.

```
procedure SavePaletteToIni(Ini: TCustomIniFile); overload;
```

**1.17.1.6.1.24.2 TPackageMng.SavePaletteToIni Method (string)**

Saves component palette settings to INI file.

```
procedure SavePaletteToIni(const FileName: string); overload;
```

**Description****1.17.1.6.1.25 TPackageMng.SaveRegInfo Method**

Saves information about installed packages and component palette in the registry. Returns True if saving was successful.

```
function SaveRegInfo(const RegKey: string = ''): Boolean;
```

**Description**

RegKey is a registry key.

RootKey is always HKEY\_LOCAL\_MACHINE.

### 1.17.1.6.1.26 TPackageMng.SetComponentOrder Method

Moves component information to specified position in palette.

```
procedure SetComponentOrder(Cmp: TComponentClassInfo; Index: integer);
```

## 1.17.1.6.2 TPackageMng Properties

### 1.17.1.6.2.1 TPackageMng.AutoSave Property

Specifies whether packages information should be saved automatically.

```
property AutoSave: Boolean;
```

#### Description

### 1.17.1.6.2.2 TPackageMng.ComponentCount Property

Specifies number of registered component classes.

```
property ComponentCount: integer;
```

#### Description

Use ComponentCount to determine number of registered component classes.

Read-only property.

### 1.17.1.6.2.3 TPackageMng.Components Property

Specifies indexed access to registered components.

```
property Components [Index: integer]: TComponentClassInfo;
```

#### Description

Use Components to access registered component via its Index. First registered component has Index = 0 and so on.

Read-only property.

### 1.17.1.6.2.4 TPackageMng.FrameInfos Property

Specifies description list of registered component classes.

```
property FrameInfos: TObjectList;
```

#### Description

Use FrameInfos to access description list of registered frames. Each element of list is an instance of TFrameInfo (see page 245).

Read-only property.

### 1.17.1.6.2.5 TPackageMng.Packages Property

Specifies list of loaded packages.

```
property Packages: TObjectList;
```

**Description**

Read-only property

**1.17.1.6.2.6 TPackageMng.Pages Property**

Specifies list of palette pages.

**property** Pages: TStrings;

**Description**

Read Pages to use list of palette pages.

Write new value for Pages to change.

**1.17.1.6.2.7 TPackageMng.RegSubkey Property**

Specifies registry key to which packages information will be saved\loaded.

**property** RegSubkey: String;

**Description****1.17.1.6.3 TPackageMng Events****1.17.1.6.3.1 TPackageMng.OnRegisterComponent Event**

Occurs before component will be registered.

**property** OnRegisterComponent: TComponentRegEvent;

**Description**

Using this event you may filter components during loading packages.

The AClass parameter is the component class is being registered currently.

The Page parameter is the name of palette page component is being installed to. Change this property to customize output.

The Accept parameter determines whether a component is allowed to be registered.

**1.17.1.6.3.2 TPackageMng.OnRegisterComponentInfo Event**

Occurs when new component class is registered in component palette.

**property** OnRegisterComponentInfo: TComponentRegInfoEvent;

**1.17.1.6.3.3 TPackageMng.OnUnRegisterComponentInfo Event**

Occurs when component information is unregistered, i.e. removed from component palette.



**property** OnUnRegisterComponentInfo: TComponentRegInfoEvent;

---

**1.17.2 Functions**

The following table lists functions in this documentation.

Functions

Function	Description
 DrawBtnIcon (  see page 257)	Draws component icon.

Legend

	Method
-----------------------------------------------------------------------------------	--------

### 1.17.2.1 ed\_RegComps.DrawBtnIcon Function

```
procedure DrawBtnIcon(Canvas: TCanvas; ARect: TRect; Icon: TBitmap; Style: TIconBtnStyle);
```

File

ed\_RegComps


Description

Draws component icon.

## 1.17.3 Structs, Records, Enums

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

Enumerations

Enumeration	Description
 TIconBtnStyle (  see page 257)	TIconBtnStyle determines style of the component icon rendering

Legend

	Enumeration
-------------------------------------------------------------------------------------	-------------

### 1.17.3.1 ed\_RegComps.TIconBtnStyle Enumeration

TIconBtnStyle determines style of the component icon rendering

```
TIconBtnStyle = (  
    ibsNone,  
    ibsPush,  
    ibsFlat  
);
```

File

ed\_RegComps

Members

Members	Description
ibsNone	no frame will be added to button
ibsPush	frame looks like standard button
ibsFlat	frame has flat border

Description

TIconBtnStyle determines how component icons will be rendered on the component palette

- ibsNone - no frame will be added to button
- ibsPush - frame looks like standard button
- ibsFlat - frame has flat border

# 1.17.4 Types

The following table lists types in this documentation.

## Types

Type	Description
TComponentRegEvent (see page 258)	See TPackageMng.OnRegisterComponent Event (see page 256)
TComponentRegInfoEvent (see page 258)	See TPackageMng.OnRegisterComponentInfo Event (see page 256)

## 1.17.4.1 ed\_RegComps.TComponentRegEvent Type

TComponentRegEvent = **procedure** (AClass: TComponentClass; **var** Page: **string**; **var** Accept: Boolean) **of object**;

### File

ed\_RegComps

### Description

See TPackageMng.OnRegisterComponent Event (see page 256)

## 1.17.4.2 ed\_RegComps.TComponentRegInfoEvent Type

See TPackageMng.OnRegisterComponentInfo Event (see page 256)

TComponentRegInfoEvent = **procedure** (AComponentClassInfo: TComponentClassInfo) **of object**;

### File

ed\_RegComps

# 1.17.5 Variables

The following table lists variables in this documentation.

## Variables

Variable	Description
PackageMng (see page 258)	Main registration object.
Runtime (see page 259)	Specifies running mode of the package. Runtime is True when package is loaded outside of Delphi IDE. You may use this variable to make some features of property or component editors to be available only in EControl Form Designer Pro.

## 1.17.5.1 ed\_RegComps.PackageMng Variable

Main registration object.

PackageMng: TPackageMng;

### File

ed\_RegComps

### Description

Use this object to access registration information of registered design objects: component editors, property editors, custom

modules.

Use this object to install/uninstall packages and to customize component palette.

Do not create objects of TPackageMng (see page 248) class. Use global variable PackageMng to access registration information.

### 1.17.5.2 ed\_RegComps.Runtime Variable

```
Runtime: Boolean = False;
```

**File**

ed\_RegComps

**Description**

Specifies running mode of the package. Runtime is True when package is loaded outside of Delphi IDE. You may use this variable to make some features of property or component editors to be available only in EControl Form Designer Pro.

## 1.18 ed\_RegMeth Namespace

### 1.18.1 Classes

The following table lists classes in this documentation.

**Classes**

Class	Description
TDefaultMethodRegister (see page 259)	Class for registration object methods in application.

#### 1.18.1.1 TDefaultMethodRegister Class

Class for registration object methods in application.

**Class Hierarchy**



```
TDefaultMethodRegister = class;
```

**File**

ed\_RegMeth

**Description**

Class for methods registration. Global variable MethRegister (see page 267): TDefaultMethodRegister is used by the Designer to assign method to procedural property (event).

To register method use AddMethod (see page 264) function.

But AddMethod (see page 264) is not comfortable to use, so class have list of overload helper-methods to simplify registration:



- Add (🔗 see page 261)(ev: TNotifyEvent)
- Add (🔗 see page 261)(ev: TMouseEvent)
- Add (🔗 see page 261)(ev: TKeyEvent)
- and so on

To simplify another types of method add (🔗 see page 261) helper function (see example)

Do not instance object of this class. Instead use global variable existing MethRegister (🔗 see page 267).

### Example

See example (🔗 see page 10) how to register methods with and without helper function.

See demo application in Demo/reg\_new\_method\_type/ too.

### Members

#### TDefaultMethodRegister Fields

TDefaultMethodRegister Fields	Description
🔗 FInfos (🔗 see page 261)	List to store PMethodInfo (🔗 see page 266) for registered methods

#### TDefaultMethodRegister Methods

TDefaultMethodRegister Methods	Description
🔗 Add (🔗 see page 261)	Use of the Add(...) methods to register existent procedure to be available in object inspector at runtime.
🔗 AddMethod (🔗 see page 264)	Register's method for using in helper-functions
🔗 Create (🔗 see page 264)	Creates and initializes the MethRegister (🔗 see page 267) object
🔗 V Destroy (🔗 see page 264)	Destroys MethRegister (🔗 see page 267) object
🔗 GetMethodsNames (🔗 see page 264)	Calls Proc for each registered method of the specified type TypeData
🔗 RemoveObject (🔗 see page 265)	Remove all entries of registered object methods
🔗 ValidateMethod (🔗 see page 265)	Looks for the method in registered methods list, then validates type of the method.

#### TDefaultMethodRegister Properties

TDefaultMethodRegister Properties	Description
🔗 R Count (🔗 see page 265)	Specifies the number of items in the array of registered methods.
🔗 R Items (🔗 see page 265)	Provides indexed access to the items in the PMethodInfo (🔗 see page 266) collection.

### Legend





🔗	Data Member
🔗	protected
🔗	Method
V	virtual
🔗	Property
R	read only

#### TDefaultMethodRegister Fields






TDefaultMethodRegister Fields	Description
🔗 FInfos (🔗 see page 261)	List to store PMethodInfo (🔗 see page 266) for registered methods

#### TDefaultMethodRegister Methods

TDefaultMethodRegister Methods	Description
🔗 Add (🔗 see page 261)	Use of the Add(...) methods to register existent procedure to be available in object inspector at runtime.
🔗 AddMethod (🔗 see page 264)	Register's method for using in helper-functions
🔗 Create (🔗 see page 264)	Creates and initializes the MethRegister (🔗 see page 267) object
🔗 V Destroy (🔗 see page 264)	Destroys MethRegister (🔗 see page 267) object
🔗 GetMethodsNames (🔗 see page 264)	Calls Proc for each registered method of the specified type TypeData


 RemoveObject (  see page 265)	Remove all entries of registered object methods
 ValidateMethod (  see page 265)	Looks for the method in registered methods list, then validates type of the method.

**TDefaultMethodRegister Properties**

TDefaultMethodRegister Properties	Description
 Count (  see page 265)	Specifies the number of items in the array of registered methods.
 Items (  see page 265)	Provides indexed access to the items in the PMethodInfo (  see page 266) collection.


**1.18.1.1.1 TDefaultMethodRegister Fields**

**1.18.1.1.1.1 TDefaultMethodRegister.FInfos Field**

List to store PMethodInfo ( see page 266) for registered methods

FInfos: TList;

**Description**

This list is used internally to store PMethodInfo ( see page 266) for registered methods.

**1.18.1.1.2 TDefaultMethodRegister Methods**

**1.18.1.1.2.1 Add Method**

Use of of the Add(...) methods to register existent procedure to be available in object inspector at runtime.

**Legend**

	Method
-------------------------------------------------------------------------------------	--------

**1.18.1.1.2.1.1 TDefaultMethodRegister.Add Method (TCanResizeEvent)**

Helper-method for TCanResizeEvent type registration

**procedure** Add(ev: TCanResizeEvent); **overload**;

**Description**

Use this method to register methods of TCanResizeEvent type.

**1.18.1.1.2.1.2 TDefaultMethodRegister.Add Method (TConstrainedResizeEvent)**

Helper-method for TConstrainedResizeEvent type registration

**procedure** Add(ev: TConstrainedResizeEvent); **overload**;

**Description**

Use this method to register methods of TConstrainedResizeEvent type.

**1.18.1.1.2.1.3 TDefaultMethodRegister.Add Method (TContextPopupEvent)**

Helper-method for TContextPopupEvent type registration

**procedure** Add(ev: TContextPopupEvent); **overload**;

**Description**

Use this method to register methods of TContextPopupEvent type.

**1.18.1.1.2.1.4 TDefaultMethodRegister.Add Method (TDockDropEvent)**

Helper-method for TDockDropEvent type registration

```
procedure Add(ev: TDockDropEvent); overload;
```

**Description**

Use this method to register methods of TDockDropEvent type.

**1.18.1.1.2.1.5 TDefaultMethodRegister.Add Method (TDockOverEvent)**

Helper-method for TDockOverEvent type registration

```
procedure Add(ev: TDockOverEvent); overload;
```

**Description**

Use this method to register methods of TDockOverEvent type.

**1.18.1.1.2.1.6 TDefaultMethodRegister.Add Method (TDragDropEvent)**

Helper-method for TDragDropEvent type registration

```
procedure Add(ev: TDragDropEvent); overload;
```

**Description**

Use this method to register methods of TDragDropEvent type.

**1.18.1.1.2.1.7 TDefaultMethodRegister.Add Method (TDragOverEvent)**

Helper-method for TDragOverEvent type registration

```
procedure Add(ev: TDragOverEvent); overload;
```

**Description**

Use this method to register methods of TDragOverEvent type.

**1.18.1.1.2.1.8 TDefaultMethodRegister.Add Method (TEndDragEvent)**

Helper-method for TEndDragEvent type registration

```
procedure Add(ev: TEndDragEvent); overload;
```

**Description**

Use this method to register methods of TEndDragEvent type.

**1.18.1.1.2.1.9 TDefaultMethodRegister.Add Method (TGetSiteInfoEvent)**

Helper-method for TGetSiteInfoEvent type registration

```
procedure Add(ev: TGetSiteInfoEvent); overload;
```

**Description**

Use this method to register methods of TGetSiteInfoEvent type.

**1.18.1.1.2.1.10 TDefaultMethodRegister.Add Method (TKeyEvent)**

Helper-method for TKeyEvent type registration

```
procedure Add(ev: TKeyEvent); overload;
```

**Description**

Use this method to register methods of TKeyEvent type.

**1.18.1.1.2.1.11 TDefaultMethodRegister.Add Method (TKeyPressEvent)**

Helper-method for TKeyPressEvent type registration

```
procedure Add(ev: TKeyPressEvent); overload;
```

**Description**

Use this method to register methods of TKeyPressEvent type.

**1.18.1.1.2.1.12 TDefaultMethodRegister.Add Method (TMouseEvent)**

Helper-method for TMouseEvent type registration

```
procedure Add(ev: TMouseEvent); overload;
```

**Description**

Use this method to register methods of TMouseEvent type.

**1.18.1.1.2.1.13 TDefaultMethodRegister.Add Method (TMouseMoveEvent)**

Helper-method for TMouseMoveEvent type registration

```
procedure Add(ev: TMouseMoveEvent); overload;
```

**Description**

Use this method to register methods of TMouseMoveEvent type.

**1.18.1.1.2.1.14 TDefaultMethodRegister.Add Method (TMouseWheelEvent)**

Helper-method for TMouseWheelEvent type registration

```
procedure Add(ev: TMouseWheelEvent); overload;
```

**Description**

Use this method to register methods of TMouseWheelEvent type.

**1.18.1.1.2.1.15 TDefaultMethodRegister.Add Method (TMouseWheelUpDownEvent)**

Helper-method for TMouseWheelUpDownEvent type registration

```
procedure Add(ev: TMouseWheelUpDownEvent); overload;
```

**Description**

Use this method to register methods of TMouseWheelUpDownEvent type.

**1.18.1.1.2.1.16 TDefaultMethodRegister.Add Method (TNotifyEvent)**

Helper-method for TNotifyEvent type registration

```
procedure Add(ev: TNotifyEvent); overload;
```

**Description**

Use this method to register methods of TNotifyEvent type.

**1.18.1.1.2.1.17 TDefaultMethodRegister.Add Method (TStartDockEvent)**

Helper-method for TStartDockEvent type registration

```
procedure Add(ev: TStartDockEvent); overload;
```

**Description**

Use this method to register methods of TStartDockEvent type.

**1.18.1.1.2.1.18 TDefaultMethodRegister.Add Method (TStartDragEvent)**

Helper-method for TStartDragEvent type registration

```
procedure Add(ev: TStartDragEvent); overload;
```

**Description**

Use this method to register methods of TStartDragEvent type.

**1.18.1.1.2.1.19 TDefaultMethodRegister.Add Method (TUnDockEvent)**

Helper-method for TUnDockEvent type registration

```
procedure Add(ev: TUnDockEvent); overload;
```

**Description**

Use this method to register methods of TUnDockEvent type.

**1.18.1.1.2.2 AddMethod Method****1.18.1.1.2.2.1 TDefaultMethodRegister.AddMethod Method (PTypeData, Pointer, TObject)**

Register's method for using in helper-functions

```
procedure AddMethod(TypeData: PTypeData; PProc: Pointer; Data: TObject); overload;
```

**Description**

See example (see page 10) for details

**1.18.1.1.2.2.2 TDefaultMethodRegister.AddMethod Method (PTypeData, TMethod)**

Register's method for direct using

```
procedure AddMethod(TypeData: PTypeData; const method: TMethod); overload;
```

**Description**

See example (see page 10) for details

**1.18.1.1.2.3 TDefaultMethodRegister.Create Constructor**

Creates and initializes the MethRegister (see page 267) object

```
constructor Create;
```

**Description**

Do not instance object of this class. Instead use global variable existing MethRegister (see page 267).

**1.18.1.1.2.4 TDefaultMethodRegister.Destroy Destructor**

Destroys MethRegister (see page 267) object

```
destructor Destroy; override;
```

**Description**

Do not destroy MethRegister (see page 267) manually. It destroys automatically when application terminates.

**1.18.1.1.2.5 TDefaultMethodRegister.GetMethodsNames Method**

Calls Proc for each registered method of the specified type TypeData

```
procedure GetMethodsNames(TypeData: PTypeData; const Proc: TGetStrProc; ARoot: TObject);
```

**Description**

The TypeData parameter is PTypeData for specified method

The Proc parameter is a callback procedure for creating name of this method

The ARoot parameter is the owner object.

### 1.18.1.1.2.6 TDefaultMethodRegister.RemoveObject Method

Remove all entries of registered object methods

```
procedure RemoveObject(ARoot: TObject);
```

Description

### 1.18.1.1.2.7 TDefaultMethodRegister.ValidateMethod Method

Looks for the method in registered methods list, then validates type of the method.

```
function ValidateMethod(TypeData: PTypeData; const method: TMethod): Boolean;
```

Description

Used for creating list of the appropriate methods.

## 1.18.1.1.3 TDefaultMethodRegister Properties

### 1.18.1.1.3.1 TDefaultMethodRegister.Count Property

Specifies the number of items in the array of registered methods.

```
property Count: integer;
```

Description

Read Count to determine the number of PMethodInfo (see page 266) entries in the array.

Read-only property

### 1.18.1.1.3.2 TDefaultMethodRegister.Items Property

Provides indexed access to the items in the PMethodInfo (see page 266) collection.

```
property Items [Index: integer]: PMethodInfo;
```


Description

Use Items to access individual items in the collection.

## 1.18.2 Structs, Records, Enums

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

### Records

Record	Description
 TMethodInfo (see page 265)	Internal method information record.

### Legend

	Record
-------------------------------------------------------------------------------------	--------

### 1.18.2.1 ed\_RegMeth.TMethodInfo Record

```
TMethodInfo = record
  MethodType: PTypeData;
```

```
Data: TObject;  
Code: pointer;  
end;
```

File

ed\_RegMeth

Members

Members	Description
MethodType: PTypeData;	Pointer to type information of the method.
Data: TObject;	Pointer to object instance.
Code: pointer;	Pointer to method code.

Description

Internal method information record.

### 1.18.3 Types

The following table lists types in this documentation.

Types

Type	Description
PMethod (see page 266)	This is a Pointer to TMethod type
PMethodInfo (see page 266)	PMethodInfo is a pointer to TMethodInfo (see page 265) type

#### 1.18.3.1 ed\_RegMeth.PMethod Type

This is a Pointer to TMethod type

```
PMethod = ^TMethod;
```

File

ed\_RegMeth

Description

#### 1.18.3.2 ed\_RegMeth.PMethodInfo Type

PMethodInfo is a pointer to TMethodInfo (see page 265) type

```
PMethodInfo = ^TMethodInfo;
```

File

ed\_RegMeth

Description

### 1.18.4 Variables

The following table lists variables in this documentation.

Variables

Variable	Description
MethRegister (🔗 see page 267)	You can replace MethRegister with instance of your class, derived from TDefaultMethodRegister (🔗 see page 259) to support more event types.

1.18.4.1 ed\_RegMeth.MethRegister Variable

You can replace MethRegister with instance of your class, derived from TDefaultMethodRegister (🔗 see page 259) to support more event types.

```
MethRegister: TDefaultMethodRegister;
```

File

ed\_RegMeth

1.19 ed\_ObjTree Namespace

1.19.1 Classes

The following table lists classes in this documentation.

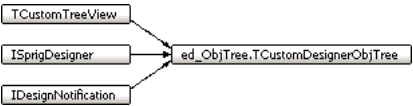
Classes

Class	Description
TCustomDesignerObjTree (🔗 see page 267)	Object TreeView displays a tree diagram of the visual and nonvisual components you place on a form, data module, or frame.
TDesignerObjTree (🔗 see page 271)	Object TreeView displays a tree diagram of the visual and nonvisual components you place on a form, data module, or frame.

1.19.1.1 TCustomDesignerObjTree Class

Object TreeView displays a tree diagram of the visual and nonvisual components you place on a form, data module, or frame.

Class Hierarchy



```
TCustomDesignerObjTree = class(TCustomTreeView, ISprigDesigner, IDesignNotification);
```

File

ed\_ObjTree

Description

Object TreeView displays the components' logical relationships, such as sibling, parent-child (such as a form to a check box), and property relationships (such as a dataset to its FieldDefs properties).Some of these relationships are implicit, such as a dataset component to its properties. You can create (🔗 see page 269) other relationships by dragging and dropping one component on top of another, if they have the possibility of a relationship. Or you can drag and drop to change the parental relationships between components.



For example, you can drag:

- Related components that you can combine, such as a panel and a check box to make a parent-child relationship.
- Data sources from one table or dataset to another.
- Databases from one session to another.
- Datasets (such as tables and queries) from one database to another.












Some nodes in the tree diagram are shown with black-and-white icons. These nodes represent "implied" components.

Double-click the item to open component editor if there is any component editor associated with selected component.




When you right-click an item in the tree diagram, you'll see an abridged version of the component's context menu. To access the full menu, right-click on the same component in the form, data module, or frame.

## Members


### TCustomDesignerObjTree Methods

TCustomDesignerObjTree Methods	Description
 AddSprigAddItems (see page 269)	Adds menu items for adding new elements.
 AddType (see page 269)	Adds item of type specified at index.
 AddTypeCount (see page 269)	Returns number of adding types.
 CanDelete (see page 269)	Determines whether selected item maybe deleted.
 CanMove (see page 269)	Determines whether item may be moved.
 Create (see page 269)	Creates and initializes a TCustomDesignerObjTree instance.
 DeleteSelected (see page 270)	Deletes selected items.
 Destroy (see page 270)	Destroys an instance of TCustomDesignerObjTree.
 Loaded (see page 270)	Initializes the component after the form file has been read into memory.
 Move (see page 270)	Moves selected item up or down.
 Notification (see page 270)	Forwards notification messages to all owned components.







### TCustomDesignerObjTree Properties

TCustomDesignerObjTree Properties	Description
 AddTypes (see page 271)	Provides indexed access to types of added items.
 Designer (see page 271)	Allows direct linking to particular designer. In this case object tree will work with objects of the Designer.Root and will work even when designer is not active.
 RootSprig (see page 271)	References root sprig item.


### TCustomDesignerObjTree Events

TCustomDesignerObjTree Events	Description
 OnCreateSprigNode (see page 271)	Called before creation node in object tree. Allows restrict sprigs to be shown and allows changing node text.












## Legend

	Method
	virtual
	protected
	Property
	read only
	Event




### TCustomDesignerObjTree Events

TCustomDesignerObjTree Events	Description
 OnCreateSprigNode (see page 271)	Called before creation node in object tree. Allows restrict sprigs to be shown and allows changing node text.

**TCustomDesignerObjTree Methods**

TCustomDesignerObjTree Methods	Description
 AddSprigAddItems (see page 269)	Adds menu items for adding new elements.
 AddType (see page 269)	Adds item of type specified at index.
 AddTypeCount (see page 269)	Returns number of adding types.
 CanDelete (see page 269)	Determines whether selected item maybe deleted.
 CanMove (see page 269)	Determines whether item may be moved.
 Create (see page 269)	Creates and initializes a TCustomDesignerObjTree instance.
 DeleteSelected (see page 270)	Deletes selected items.
 Destroy (see page 270)	Destroys an instance of TCustomDesignerObjTree.
 Loaded (see page 270)	Initializes the component after the form file has been read into memory.
 Move (see page 270)	Moves selected item up or down.
 Notification (see page 270)	Forwards notification messages to all owned components.

**TCustomDesignerObjTree Properties**

TCustomDesignerObjTree Properties	Description
 AddTypes (see page 271)	Provides indexed access to types of added items.
 Designer (see page 271)	Allows direct linking to particular designer. In this case object tree will work with objects of the Designer.Root and will work even when designer is not active.
 RootSprig (see page 271)	References root sprig item.

**1.19.1.1.1 TCustomDesignerObjTree Methods****1.19.1.1.1.1 TCustomDesignerObjTree.AddSprigAddItems Method**

Adds menu items for adding new elements.

```
procedure AddSprigAddItems(Menu: TPopupMenu);
```

**1.19.1.1.1.2 TCustomDesignerObjTree.AddType Method**

Adds item of type specified at index.

```
procedure AddType(Index: Integer);
```

**1.19.1.1.1.3 TCustomDesignerObjTree.AddTypeCount Method**

Returns number of adding types.

```
function AddTypeCount: Integer;
```

**1.19.1.1.1.4 TCustomDesignerObjTree.CanDelete Method**

Determines whether selected item maybe deleted.

```
function CanDelete: Boolean;
```

**1.19.1.1.1.5 TCustomDesignerObjTree.CanMove Method**

Determines whether item may be moved.

```
function CanMove(Up: Boolean): Boolean;
```

**1.19.1.1.1.6 TCustomDesignerObjTree.Create Constructor**

Creates and initializes a TCustomDesignerObjTree instance.

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

**Description**

Use Create to programmatically instantiate a TCustomDesignerObjTree component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

#### 1.19.1.1.1.7 TCustomDesignerObjTree.DeleteSelected Method

Deletes selected items.

```
procedure DeleteSelected; reintroduce;
```

#### 1.19.1.1.1.8 TCustomDesignerObjTree.Destroy Destructor

Destroys an instance of TCustomDesignerObjTree.

```
destructor Destroy; override;
```

##### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

#### 1.19.1.1.1.9 TCustomDesignerObjTree.Loaded Method

Initializes the component after the form file has been read into memory.

```
procedure Loaded; override;
```

##### Description

Do not call the protected Loaded method. The streaming system calls this method after it loads the component's form from a stream.

When the streaming system loads a form or data module from its form file, it first constructs the form component by calling its constructor, then reads its property values from the form file. After reading all the property values for all the components, the streaming system calls the Loaded methods of each component in the order the components were created. This gives the components a chance to initialize any data that depends on the values of other components or other parts of itself.

##### Notes

All references to sibling components are resolved by the time Loaded is called. Loaded is the first place that sibling pointers can be used after being streamed in.

As implemented in TComponent, Loaded clears the csLoading flag in the ComponentState property, indicating that the component is no longer loading.

**Warning:** Loaded may be called multiple times on inherited forms. It is called every time a level of inheritance is streamed in. Do not allocate memory in an overridden Loaded method without first checking that the memory has not been allocated in a previous call.

#### 1.19.1.1.1.10 TCustomDesignerObjTree.Move Method

Moves selected item up or down.

```
procedure Move(Up: Boolean);
```

#### 1.19.1.1.1.11 TCustomDesignerObjTree.Notification Method

Forwards notification messages to all owned components.

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

##### Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by

AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

#### Notes

Notification is not called for components that are freed implicitly (because their Owner is freed).

### 1.19.1.1.2 TCustomDesignerObjTree Properties

#### 1.19.1.1.2.1 TCustomDesignerObjTree.AddTypes Property

Provides indexed access to types of added items.

```
property AddTypes [Index: Integer]: string;
```

#### 1.19.1.1.2.2 TCustomDesignerObjTree.Designer Property

Allows direct linking to particular designer. In this case object tree will work with objects of the Designer.Root and will work even when designer is not active.

```
property Designer: TzCustomFormDesigner;
```

#### 1.19.1.1.2.3 TCustomDesignerObjTree.RootSprig Property

References root sprig item.

```
property RootSprig: TRootSprig;
```

### 1.19.1.1.3 TCustomDesignerObjTree Events

#### 1.19.1.1.3.1 TCustomDesignerObjTree.OnCreateSprigNode Event

Called before creation node in object tree. Allows restrict sprigs to be shown and allows changing node text.

```
property OnCreateSprigNode: TCreateSprigNodeEvent;
```

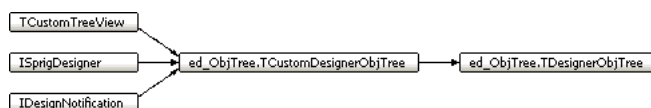
#### Notes

Works only in BDS 2005 and later versions.

### 1.19.1.2 TDesignerObjTree Class

Object TreeView displays a tree diagram of the visual and nonvisual components you place on a form, data module, or frame.

#### Class Hierarchy



```
TDesignerObjTree = class(TCustomDesignerObjTree);
```

#### File

ed\_ObjTree

## Description

Object TreeView displays the components' logical relationships, such as sibling, parent-child (such as a form to a check box), and property relationships (such as a dataset to its FieldDefs properties). Some of these relationships are implicit, such as a dataset component to its properties. You can create (see page 277) other relationships by dragging and dropping one component on top of another, if they have the possibility of a relationship. Or you can drag and drop to change the parental relationships between components.

For example, you can drag:

- Related components that you can combine, such as a panel and a check box to make a parent-child relationship.
- Data sources from one table or dataset to another.
- Databases from one session to another.
- Datasets (such as tables and queries) from one database to another.












Some nodes in the tree diagram are shown with black-and-white icons. These nodes represent "implied" components.

Double-click the item to open component editor if there is any component editor associated with selected component.


When you right-click an item in the tree diagram, you'll see an abridged version of the component's context menu. To access the full menu, right-click on the same component in the form, data module, or frame.

## Members




### TCustomDesignerObjTree Methods

TCustomDesignerObjTree Methods	Description
 AddSprigAddItems (see page 269)	Adds menu items for adding new elements.
 AddType (see page 269)	Adds item of type specified at index.
 AddTypeCount (see page 269)	Returns number of adding types.
 CanDelete (see page 269)	Determines whether selected item maybe deleted.
 CanMove (see page 269)	Determines whether item may be moved.
 Create (see page 269)	Creates and initializes a TCustomDesignerObjTree instance.
 DeleteSelected (see page 270)	Deletes selected items.
 Destroy (see page 270)	Destroys an instance of TCustomDesignerObjTree.
 Loaded (see page 270)	Initializes the component after the form file has been read into memory.
 Move (see page 270)	Moves selected item up or down.
 Notification (see page 270)	Forwards notification messages to all owned components.





### TDesignerObjTree Class

































TDesignerObjTree Class	Description
 Create (see page 277)	Creates and initializes a TDesignerObjTree instance.

### TCustomDesignerObjTree Properties

TCustomDesignerObjTree Properties	Description
 AddTypes (see page 271)	Provides indexed access to types of added items.
 Designer (see page 271)	Allows direct linking to particular designer. In this case object tree will work with objects of the Designer.Root and will work even when designer is not active.
 RootSprig (see page 271)	References root sprig item.


### TDesignerObjTree Class

TDesignerObjTree Class	Description
 Align (see page 277)	Determines how the control aligns within its container (parent control).
 Anchors (see page 277)	Specifies how the control is anchored to its parent.
 AutoExpand (see page 278)	Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.
 BevelEdges (see page 278)	Specifies which edges of the control are beveled.







 BevelInner (see page 278)	Specifies the cut of the inner bevel.
 BevelKind (see page 278)	Specifies the control's bevel style.
 BevelOuter (see page 279)	Specifies the cut of the outer bevel.
 BevelWidth (see page 279)	Specifies the width of the inner and outer bevels.
 BiDiMode (see page 279)	Specifies the bi-directional mode for the control.
 BorderStyle (see page 279)	Determines whether the tree view control has a border.
 BorderWidth (see page 279)	Specifies the width of the control's border.
 ChangeDelay (see page 280)	Specifies the delay between when a node is selected and when the OnChange event occurs.
 Color (see page 280)	Specifies the background color of the control.
 Constraints (see page 280)	Specifies the size constraints for the control.
 Ctl3D (see page 280)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DragCursor (see page 280)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 281)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 281)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 281)	Controls whether the control responds to mouse, keyboard, and timer events.
 Font (see page 281)	Controls the attributes of text written on or in the control.
 HideSelection (see page 281)	Determines whether a selected node appears selected when the focus shifts to another control.
 HotTrack (see page 281)	Specifies whether list items are highlighted when the mouse passes over them.
 Images (see page 282)	Determines which image list is associated with the tree view.
 Indent (see page 282)	Specifies the amount of indentation in pixels when a list of child nodes is expanded.
 Items (see page 282)	Lists the individual nodes that appear in the tree view control.
 MultiSelect (see page 282)	Determines whether the user can select more than one tree node at a time.
 MultiSelectStyle (see page 282)	Determines how multiple node selections work.
 OnAddition (see page 283)	Occurs when new node is added.
 OnAdvancedCustomDraw (see page 283)	Occurs at discrete stages during the painting of the tree view control.
 OnAdvancedCustomDrawItem (see page 283)	Occurs at discrete stages during the painting of tree view nodes.
 OnChange (see page 283)	Occurs whenever the selection has changed from one node to another.
 OnChanging (see page 284)	Occurs when the selection is about to change from one node to another.
 OnClick (see page 284)	Occurs when the user clicks the control.
 OnCollapsed (see page 284)	Occurs after a node has been collapsed.
 OnCollapsing (see page 284)	Occurs when a node is about to be collapsed.
 OnCompare (see page 284)	Occurs when two nodes must be compared during a sort of the nodes in the tree view.
 OnContextPopup (see page 285)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnCreateNodeClass (see page 285)	Occurs when a node in the treeview is about to be created.
 OnCreateSprigNode (see page 285)	Called before creation node in object tree. Allows restrict sprigs to be shown and allows changing node text.
 OnCustomDraw (see page 285)	Occurs immediately prior to painting the tree view control.
 OnCustomDrawItem (see page 286)	Occurs immediately prior to painting a node in a tree view control.
 OnDbClick (see page 286)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDeletion (see page 286)	Occurs when a node in the tree view is deleted.
 OnDragDrop (see page 286)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 286)	Occurs when the user drags an object over a control.
 OnEdited (see page 287)	Occurs after the user edits the Text property of a node.
 OnEditing (see page 287)	Occurs when the user starts to edit the Text property of a node.
 OnEndDock (see page 287)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
 OnEndDrag (see page 287)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 288)	Occurs when a control receives the input focus.
 OnExit (see page 288)	Occurs when the input focus shifts away from one control to another.
 OnExpanded (see page 288)	Occurs after a node is expanded.
 OnExpanding (see page 288)	Occurs when a node is about to be expanded.
 OnGetImageIndex (see page 288)	Occurs when the tree view looks up the ImageIndex of a node.

 OnGetSelectedIndex (see page 288)	Occurs when the tree view looks up the SelectedIndex of a node.
 OnKeyDown (see page 289)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 289)	Occurs when key pressed.
 OnKeyUp (see page 289)	Occurs when the user releases a key that has been pressed.
 OnMouseDown (see page 289)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 290)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 290)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnStartDock (see page 290)	Occurs when the user begins to drag a control with a DragKind of dkDock.
 OnStartDrag (see page 291)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 291)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 291)	Determines where a control looks for its color information.
 ParentCtl3D (see page 291)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 291)	Determines where a control looks for its font information.
 ParentShowHint (see page 291)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 292)	Identifies the pop-up menu associated with the control.
 ReadOnly (see page 292)	Determines whether the user can edit the node labels.
 RightClickSelect (see page 292)	Determines whether the Selected property returns nodes that are selected using the right mouse button.
 RowSelect (see page 292)	Specifies whether the entire row of the selected item is highlighted.
 ShowButtons (see page 292)	Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.
 ShowHint (see page 293)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 ShowLines (see page 293)	Specifies whether to display the lines that link child nodes to their corresponding parent nodes.
 ShowRoot (see page 293)	Specifies whether lines connecting top-level nodes are displayed.
 SortType (see page 293)	Determines if and how the nodes in a tree view are automatically sorted.
 StateImages (see page 293)	Determines which image list to use for state images.
 TabOrder (see page 293)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 293)	Determines if the user can tab to a control.
 ToolTips (see page 294)	Specifies whether the items in the tree view have tool tips.
 Visible (see page 294)	Determines whether the component appears on screen.


### TCustomDesignerObjTree Events

TCustomDesignerObjTree Events	Description
 OnCreateSprigNode (see page 271)	Called before creation node in object tree. Allows restrict sprigs to be shown and allows changing node text.





### Legend















	Method
	virtual
	protected
	Property
	read only
	Event

### TCustomDesignerObjTree Events



TCustomDesignerObjTree Events	Description
 OnCreateSprigNode (see page 271)	Called before creation node in object tree. Allows restrict sprigs to be shown and allows changing node text.

### TCustomDesignerObjTree Methods







TCustomDesignerObjTree Methods	Description
 AddSprigAddItems (see page 269)	Adds menu items for adding new elements.
 AddType (see page 269)	Adds item of type specified at index.
 AddTypeCount (see page 269)	Returns number of adding types.
 CanDelete (see page 269)	Determines whether selected item maybe deleted.

 CanMove (  see page 269)	Determines whether item may be moved.
 Create (  see page 269)	Creates and initializes a TCustomDesignerObjTree instance.
 DeleteSelected (  see page 270)	Deletes selected items.
 Destroy (  see page 270)	Destroys an instance of TCustomDesignerObjTree.
 Loaded (  see page 270)	Initializes the component after the form file has been read into memory.
 Move (  see page 270)	Moves selected item up or down.
 Notification (  see page 270)	Forwards notification messages to all owned components.


**TDesignerObjTree Class**

TDesignerObjTree Class	Description
 Create (  see page 277)	Creates and initializes a TDesignerObjTree instance.

















































**TCustomDesignerObjTree Properties**

TCustomDesignerObjTree Properties	Description
 AddTypes (  see page 271)	Provides indexed access to types of added items.
 Designer (  see page 271)	Allows direct linking to particular designer. In this case object tree will work with objects of the Designer.Root and will work even when designer is not active.
 RootSprig (  see page 271)	References root sprig item.

**TDesignerObjTree Class**

TDesignerObjTree Class	Description
 Align (  see page 277)	Determines how the control aligns within its container (parent control).
 Anchors (  see page 277)	Specifies how the control is anchored to its parent.
 AutoExpand (  see page 278)	Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.
 BevelEdges (  see page 278)	Specifies which edges of the control are beveled.
 BevelInner (  see page 278)	Specifies the cut of the inner bevel.
 BevelKind (  see page 278)	Specifies the control's bevel style.
 BevelOuter (  see page 279)	Specifies the cut of the outer bevel.
 BevelWidth (  see page 279)	Specifies the width of the inner and outer bevels.
 BiDiMode (  see page 279)	Specifies the bi-directional mode for the control.
 BorderStyle (  see page 279)	Determines whether the tree view control has a border.
 BorderWidth (  see page 279)	Specifies the width of the control's border.
 ChangeDelay (  see page 280)	Specifies the delay between when a node is selected and when the OnChange event occurs.
 Color (  see page 280)	Specifies the background color of the control.
 Constraints (  see page 280)	Specifies the size constraints for the control.
 Ctl3D (  see page 280)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DragCursor (  see page 280)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (  see page 281)	Specifies whether the control is being dragged normally or for docking.
 DragMode (  see page 281)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (  see page 281)	Controls whether the control responds to mouse, keyboard, and timer events.
 Font (  see page 281)	Controls the attributes of text written on or in the control.
 HideSelection (  see page 281)	Determines whether a selected node appears selected when the focus shifts to another control.
 HotTrack (  see page 281)	Specifies whether list items are highlighted when the mouse passes over them.
 Images (  see page 282)	Determines which image list is associated with the tree view.
 Indent (  see page 282)	Specifies the amount of indentation in pixels when a list of child nodes is expanded.
 Items (  see page 282)	Lists the individual nodes that appear in the tree view control.
 MultiSelect (  see page 282)	Determines whether the user can select more than one tree node at a time.
 MultiSelectStyle (  see page 282)	Determines how multiple node selections work.
 OnAddition (  see page 283)	Occurs when new node is added.
 OnAdvancedCustomDraw (  see page 283)	Occurs at discrete stages during the painting of the tree view control.
 OnAdvancedCustomDrawItem (  see page 283)	Occurs at discrete stages during the painting of tree view nodes.
 OnChange (  see page 283)	Occurs whenever the selection has changed from one node to another.
 OnChanging (  see page 284)	Occurs when the selection is about to change from one node to another.
 OnClick (  see page 284)	Occurs when the user clicks the control.



 OnCollapsed (see page 284)	Occurs after a node has been collapsed.
 OnCollapsing (see page 284)	Occurs when a node is about to be collapsed.
 OnCompare (see page 284)	Occurs when two nodes must be compared during a sort of the nodes in the tree view.
 OnContextPopup (see page 285)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnCreateNodeClass (see page 285)	Occurs when a node in the treeview is about to be created.
 OnCreateSprigNode (see page 285)	Called before creation node in object tree. Allows restrict sprigs to be shown and allows changing node text.
 OnCustomDraw (see page 285)	Occurs immediately prior to painting the tree view control.
 OnCustomDrawItem (see page 286)	Occurs immediately prior to painting a node in a tree view control.
 OnDbClick (see page 286)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDeletion (see page 286)	Occurs when a node in the tree view is deleted.
 OnDragDrop (see page 286)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 286)	Occurs when the user drags an object over a control.
 OnEdited (see page 287)	Occurs after the user edits the Text property of a node.
 OnEditing (see page 287)	Occurs when the user starts to edit the Text property of a node.
 OnEndDock (see page 287)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
 OnEndDrag (see page 287)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 288)	Occurs when a control receives the input focus.
 OnExit (see page 288)	Occurs when the input focus shifts away from one control to another.
 OnExpanded (see page 288)	Occurs after a node is expanded.
 OnExpanding (see page 288)	Occurs when a node is about to be expanded.
 OnGetImageIndex (see page 288)	Occurs when the tree view looks up the ImageIndex of a node.
 OnGetSelectedIndex (see page 288)	Occurs when the tree view looks up the SelectedIndex of a node.
 OnKeyDown (see page 289)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 289)	Occurs when key pressed.
 OnKeyUp (see page 289)	Occurs when the user releases a key that has been pressed.
 OnMouseDown (see page 289)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 290)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 290)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnStartDock (see page 290)	Occurs when the user begins to drag a control with a DragKind of dkDock.
 OnStartDrag (see page 291)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 291)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 291)	Determines where a control looks for its color information.
 ParentCtl3D (see page 291)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 291)	Determines where a control looks for its font information.
 ParentShowHint (see page 291)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 292)	Identifies the pop-up menu associated with the control.
 ReadOnly (see page 292)	Determines whether the user can edit the node labels.
 RightClickSelect (see page 292)	Determines whether the Selected property returns nodes that are selected using the right mouse button.
 RowSelect (see page 292)	Specifies whether the entire row of the selected item is highlighted.
 ShowButtons (see page 292)	Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.
 ShowHint (see page 293)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 ShowLines (see page 293)	Specifies whether to display the lines that link child nodes to their corresponding parent nodes.
 ShowRoot (see page 293)	Specifies whether lines connecting top-level nodes are displayed.
 SortType (see page 293)	Determines if and how the nodes in a tree view are automatically sorted.
 StateImages (see page 293)	Determines which image list to use for state images.
 TabOrder (see page 293)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 293)	Determines if the user can tab to a control.
 ToolTips (see page 294)	Specifies whether the items in the tree view have tool tips.

 Visible (see page 294)

Determines whether the component appears on screen.

## 1.19.1.2.1 TDesignerObjTree Methods

### 1.19.1.2.1.1 TDesignerObjTree.Create Constructor

Creates and initializes a TDesignerObjTree instance.

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

#### Description

Use Create to programmatically instantiate a TDesignerObjTree component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

## 1.19.1.2.2 TDesignerObjTree Properties

### 1.19.1.2.2.1 TDesignerObjTree.Align Property

Determines how the control aligns within its container (parent control).

```
property Align;
```

#### Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

**Tip:** If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

**Note:** To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

### 1.19.1.2.2.2 TDesignerObjTree.Anchors Property

Specifies how the control is anchored to its parent.

```
property Anchors;
```

#### Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is

resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its `Anchors` property set to `[akLeft, akRight]`, the control stretches when the width of its parent changes.

`Anchors` is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

**Note:** If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the `Align` property instead. Unlike `Anchors`, `Align` allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

#### 1.19.1.2.2.3 TDesignerObjTree.AutoExpand Property

Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection.

```
property AutoExpand;
```

##### Description

Set `AutoExpand` to true to cause the selected item to expand and the unselected items to collapse.

#### 1.19.1.2.2.4 TDesignerObjTree.BevelEdges Property

Specifies which edges of the control are beveled.

```
property BevelEdges;
```

##### Description

Use `BevelEdges` to get or set which edges of the control are beveled. The `BevelInner`, `BevelOuter`, and `BevelKind` properties determine the appearance of the specified edges.

#### 1.19.1.2.2.5 TDesignerObjTree.BevelInner Property

Specifies the cut of the inner bevel.

```
property BevelInner;
```

##### Description

Use `BevelInner` to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (`BevelOuter` is `bvNone`), the inner bevel appears immediately inside the border.

#### 1.19.1.2.2.6 TDesignerObjTree.BevelKind Property

Specifies the control's bevel style.

```
property BevelKind;
```

##### Description

Use `BevelKind` to modify the appearance of a bevel. `BevelKind` influences how sharply the bevel stands out.

`BevelKind`, in combination with `BevelWidth` and the cut of the bevel specified by `BevelInner` or `BevelOuter`, can create a variety of effects. Experiment with various combinations to get the look you want.

### 1.19.1.2.2.7 TDesignerObjTree.BevelOuter Property

Specifies the cut of the outer bevel.

```
property BevelOuter;
```

#### Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

### 1.19.1.2.2.8 TDesignerObjTree.BevelWidth Property

Specifies the width of the inner and outer bevels.

```
property BevelWidth;
```

#### Description

Use BevelWidth to specify the width, in pixels, of the inner and outer bevels.

### 1.19.1.2.2.9 TDesignerObjTree.BiDiMode Property

Specifies the bi-directional mode for the control.

```
property BiDiMode;
```

#### Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

### 1.19.1.2.2.10 TDesignerObjTree.BorderStyle Property

Determines whether the tree view control has a border.

```
property BorderStyle;
```

#### Description

Set BorderStyle to specify whether the tree view control should be outlined with a single-line border. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

### 1.19.1.2.2.11 TDesignerObjTree.BorderWidth Property

Specifies the width of the control's border.

```
property BorderWidth;
```

#### Description

Use BorderWidth to get or set the width of the control's border. Graphics or text drawn by the control is clipped to the area within the border.

#### 1.19.1.2.2.12 TDesignerObjTree.ChangeDelay Property

Specifies the delay between when a node is selected and when the OnChange event occurs.

**property** ChangeDelay;

##### Description

Use ChangeDelay to get or set the delay, in milliseconds, between when a node is selected and when the OnChange event occurs.

Set the ChangeDelay to 50 milliseconds to emulate the behavior of the tree-view control used in Windows Explorer.

#### 1.19.1.2.2.13 TDesignerObjTree.Color Property

Specifies the background color of the control.

**property** Color;

##### Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

#### 1.19.1.2.2.14 TDesignerObjTree.Constraints Property

Specifies the size constraints for the control.

**property** Constraints;

##### Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

**Warning:** Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

#### 1.19.1.2.2.15 TDesignerObjTree.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

**property** Ctl3D;

##### Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

#### 1.19.1.2.2.16 TDesignerObjTree.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

**property** DragCursor;

##### Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

**Note:** To make a custom cursor available for the DragCursor property, see the Cursor property.

### 1.19.1.2.2.17 TDesignerObjTree.DragKind Property

Specifies whether the control is being dragged normally or for docking.

```
property DragKind;
```

#### Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

### 1.19.1.2.2.18 TDesignerObjTree.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

```
property DragMode;
```

#### Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

### 1.19.1.2.2.19 TDesignerObjTree.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

```
property Enabled;
```

#### Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

### 1.19.1.2.2.20 TDesignerObjTree.Font Property

Controls the attributes of text written on or in the control.

```
property Font;
```

#### Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

### 1.19.1.2.2.21 TDesignerObjTree.HideSelection Property

Determines whether a selected node appears selected when the focus shifts to another control.

```
property HideSelection;
```

#### Description

Use HideSelection to specify whether the user is given visual feedback about the current selection in the tree view when it does not have focus. If true, the selected node is not visually distinct from other nodes until focus returns to the control. If false, the node always appears selected.

### 1.19.1.2.2.22 TDesignerObjTree.HotTrack Property

Specifies whether list items are highlighted when the mouse passes over them.

```
property HotTrack;
```

#### Description

Set HotTrack to true to provide visual feedback about which item is under the mouse. Set HotTrack to false to suppress the

visual feedback about which item is under the mouse.

#### 1.19.1.2.2.23 TDesignerObjTree.Images Property

Determines which image list is associated with the tree view.

```
property Images;
```

##### Description

Use Images to provide a customized list of bitmaps that can be displayed to the left of a node's label. Individual nodes specify the image from this list that should appear by setting their ImageIndex property.

#### 1.19.1.2.2.24 TDesignerObjTree.Indent Property

Specifies the amount of indentation in pixels when a list of child nodes is expanded.

```
property Indent;
```

##### Description

Use Indent to determine how far child nodes are indented from their parent nodes when the parent is expanded.

#### 1.19.1.2.2.25 TDesignerObjTree.Items Property

Lists the individual nodes that appear in the tree view control.

```
property Items;
```

##### Description

Individual nodes in a tree view are TTreeNode objects. These individual nodes can be accessed by using the Items property along with the item's index into the tree view. For example, to access the second item in the tree view, you could use the following code.

```
MyTreeNode := TreeView1.Items[1];
```

When setting this property at design-time in the Object Inspector the Tree View Items Editor appears. Use the New Item and New SubItem buttons to add items to the tree view. Use the Text property to modify what text is displayed in the label of the item.

At run-time nodes can be added and inserted by using the TTreeNodes methods AddChildFirst, AddChild, AddChildObjectFirst, AddChildObject, AddFirst, Add, AddObjectFirst, AddObject, and Insert.

#### 1.19.1.2.2.26 TDesignerObjTree.MultiSelect Property

Determines whether the user can select more than one tree node at a time.

```
property MultiSelect;
```

##### Description

Set MultiSelect to specify whether users can select multiple nodes using the Control and Shift keys. A selection style must also be chosen in MultiSelectStyle.

#### 1.19.1.2.2.27 TDesignerObjTree.MultiSelectStyle Property

Determines how multiple node selections work.

```
property MultiSelectStyle;
```

**Description**

MultiSelectStyle determines how multiple selections are made when MultiSelect is true. MultiSelectStyle must include at least one of the following values.

Value	Meaning
msControlSelect	Clicking on any node with the Control key pressed toggles the selection of that node.
msShiftSelect	Clicking on any node with the Shift key press selects that node, the last single node selected, and the nodes in between. All other nodes are deselected.
msVisibleOnly	Multiple selections with the Shift key do not include child nodes of collapsed nodes.
msSiblingOnly	Selected nodes are restricted to a single set of siblings.

If msControlSelect or msShiftSelect are in effect, the last singly-selected node becomes the primary selection, referenced by Selections[0]. The primary selection is the anchor for extended selections using the Shift key.

**1.19.1.2.2.28 TDesignerObjTree.OnAddition Property**

Occurs when new node is added.

**property** OnAddition;

**Description**

OnAddition occurs when a new node is added to the control.

**1.19.1.2.2.29 TDesignerObjTree.OnAdvancedCustomDraw Property**

Occurs at discrete stages during the painting of the tree view control.

**property** OnAdvancedCustomDraw;

**Description**

Write an OnAdvancedCustomDraw event handler to paint an owner-drawn tree view. Use the Canvas property as a drawing surface when painting the image of the tree view.

To paint individual items, use the OnAdvancedCustomDrawItem or OnCustomDrawItem event instead.

**Notes**

OnAdvancedCustomDraw occurs at several stages during the paint process, not just immediately prior to the default rendering. If you only need to use the cdPrePaint stage, it is more efficient to use the OnCustomDraw event.

**1.19.1.2.2.30 TDesignerObjTree.OnAdvancedCustomDrawItem Property**

Occurs at discrete stages during the painting of tree view nodes.

**property** OnAdvancedCustomDrawItem;

**Description**

Write an OnAdvancedCustomDrawItem event handler to customize the painting of individual items in the tree view.

**Notes**

OnAdvancedCustomDrawItem occurs at several stages during the paint process and allows you to suppress the default painting of tree node images while still allowing the default painting of the node. If you only need to use the cdPrePaint stage and do not want to suppress only the painting of tree node images, it is more efficient to use the OnCustomDrawItem event.

**1.19.1.2.2.31 TDesignerObjTree.OnChange Property**

Occurs whenever the selection has changed from one node to another.



```
property OnChange;
```

**Description**

Write an OnChange event handler to take specific action when the selected node changes. The Sender parameter is the tree view whose selected node changes, and the Node parameter is the newly selected node.

**Notes**

The OnChange event does not occur for nodes selected using the right mouse button when RightClickSelect is true. To respond to changes in the value of the Selected property when RightClickSelect is true, use the OnMouseUp event.

**1.19.1.2.2.32 TDesignerObjTree.OnChanging Property**

Occurs when the selection is about to change from one node to another.

```
property OnChanging;
```

**Description**

Write an OnChanging event handler to selectively prevent selection from moving to specific nodes.

The TTVChangingEvent type points to a method that is called when the selection is about to be changed from one node to another. Set AllowChange to false, to prevent selection from moving to a new node. The Node parameter specifies the node that is about to be selected.

**1.19.1.2.2.33 TDesignerObjTree.OnClick Property**

Occurs when the user clicks the control.

```
property OnClick;
```

**1.19.1.2.2.34 TDesignerObjTree.OnCollapsed Property**

Occurs after a node has been collapsed.

```
property OnCollapsed;
```

**Description**

Write an OnCollapsed event handler to respond after a node in the tree view collapses. The Node parameter is the node whose children are no longer visible.

**1.19.1.2.2.35 TDesignerObjTree.OnCollapsing Property**

Occurs when a node is about to be collapsed.

```
property OnCollapsing;
```

**Description**

The TTVCollapsingEvent type points to a method that is called when a node is about to be collapsed. Set the AllowCollapse parameter to false to prevent the node specified by the Node parameter from being collapsed.

**1.19.1.2.2.36 TDesignerObjTree.OnCompare Property**

Occurs when two nodes must be compared during a sort of the nodes in the tree view.

```
property OnCompare;
```

**Description**

Write an OnCompare event handler to customize the sort order of the nodes in the tree view. Set the Compare parameter to a value less than 0 if Node1 is less than Node2. Set Compare to 0 if Node1 is equivalent to Node2, and set Compare to a value greater than 0 if Node1 is greater than Node2. If an OnCompare event handler is not provided, tree view nodes are sorted alphabetically, based on their labels.

### 1.19.1.2.2.37 TDesignerObjTree.OnContextPopup Property

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

```
property OnContextPopup;
```

#### Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM\_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

**Note:** Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

### 1.19.1.2.2.38 TDesignerObjTree.OnCreateNodeClass Property

Occurs when a node in the treeview is about to be created.

```
property OnCreateNodeClass;
```

#### Description

OnCreateNodeClass occurs when a new node object is about to be created.

Sender is the tree view object that is about to add a new node.

NodeClass returns a class reference that is instantiated to create a new node object.

### 1.19.1.2.2.39 TDesignerObjTree.OnCreateSprigNode Property

Called before creation node in object tree. Allows restrict sprigs to be shown and allows changing node text.

```
property OnCreateSprigNode: TCreateSprigNodeEvent;
```

#### Notes

Works only in BDS 2005 and later versions.

### 1.19.1.2.2.40 TDesignerObjTree.OnCustomDraw Property

Occurs immediately prior to painting the tree view control.

```
property OnCustomDraw;
```

**Description**

Write an OnCustomDraw event handler to paint an owner-drawn tree view. Use the Canvas property as a drawing surface when painting the image of the tree view.

To paint individual items, use the OnCustomDrawItem event instead.

**Notes**

OnCustomDraw only occurs prior to painting the tree view control. To customize the painting at other stages of the paint process (such as after the default drawing), use OnAdvancedCustomDraw instead.

**1.19.1.2.2.41 TDesignerObjTree.OnCustomDrawItem Property**

Occurs immediately prior to painting a node in a tree view control.

```
property OnCustomDrawItem;
```

**Description**

Write an OnCustomDrawItem event handler to paint individual items in the tree view, or to provide a background to the item before the default rendering of the item.

**Notes**

OnCustomDrawItem only occurs prior to painting individual tree items. To customize the painting of items at other stages of the paint process (such as after the item is painted), use OnAdvancedCustomDrawItem instead.

**1.19.1.2.2.42 TDesignerObjTree.OnDbClick Property**

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

```
property OnDbClick;
```

**Description**

Use the OnDbClick event to respond to mouse double-clicks.

**1.19.1.2.2.43 TDesignerObjTree.OnDeletion Property**

Occurs when a node in the tree view is deleted.

```
property OnDeletion;
```

**Description**

Write an OnDeletion event handler to respond when a node is deleted from the tree view control.

**1.19.1.2.2.44 TDesignerObjTree.OnDragDrop Property**

Occurs when the user drops an object being dragged.

```
property OnDragDrop;
```

**Description**

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

**1.19.1.2.2.45 TDesignerObjTree.OnDragOver Property**

Occurs when the user drags an object over a control.

```
property OnDragOver;
```

**Description**

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

**Note:** Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

**1.19.1.2.2.46 TDesignerObjTree.OnEdited Property**

Occurs after the user edits the Text property of a node.

```
property OnEdited;
```

**Description**

Write an OnEdited event handler to respond to changes the user makes to the node labels. The Node parameter is the node whose label was edited. The S parameter is the new value of the node's Text property. The node's label can be changed in an OnEdited event handler before the user's edits are committed. This event can occur only if ReadOnly is set to false.

**1.19.1.2.2.47 TDesignerObjTree.OnEditing Property**

Occurs when the user starts to edit the Text property of a node.

```
property OnEditing;
```

**Description**

Write an OnEditing event handler to determine whether the user is allowed to edit the label of a specific node in the tree view. Set the AllowEdit parameter to false to prevent the user from editing the node specified by the Node parameter. To disallow editing of all nodes in the tree view, use the ReadOnly property instead.

**1.19.1.2.2.48 TDesignerObjTree.OnEndDock Property**

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

```
property OnEndDock;
```

**Description**

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

**1.19.1.2.2.49 TDesignerObjTree.OnEndDrag Property**

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

```
property OnEndDrag;
```

**Description**

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

#### 1.19.1.2.2.50 TDesignerObjTree.OnEnter Property

Occurs when a control receives the input focus.

```
property OnEnter;
```

##### Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

#### 1.19.1.2.2.51 TDesignerObjTree.OnExit Property

Occurs when the input focus shifts away from one control to another.

```
property OnExit;
```

##### Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

#### 1.19.1.2.2.52 TDesignerObjTree.OnExpanded Property

Occurs after a node is expanded.

```
property OnExpanded;
```

##### Description

Write an OnExpanded event handler to respond when a node in the tree view is expanded. The Node parameter specifies the node whose children are now displayed to the user.

#### 1.19.1.2.2.53 TDesignerObjTree.OnExpanding Property

Occurs when a node is about to be expanded.

```
property OnExpanding;
```

##### Description

Write an OnExpanding event handler to determine whether a node can be expanded. Set the AllowExpansion parameter to false to prevent the node from expanding.

#### 1.19.1.2.2.54 TDesignerObjTree.OnGetImageIndex Property

Occurs when the tree view looks up the ImageIndex of a node.

```
property OnGetImageIndex;
```

##### Description

Write an OnGetImageIndex event handler to change the image index for the particular node before it is drawn. For example, the bitmap of a node can be changed to indicate a different state for the node.

#### 1.19.1.2.2.55 TDesignerObjTree.OnGetSelectedIndex Property

Occurs when the tree view looks up the SelectedIndex of a node.

```
property OnGetSelectedIndex;
```

##### Description

Write an OnGetSelectedIndex event handler to change the selected image index of a node before it is drawn.

### 1.19.1.2.2.56 TDesignerObjTree.OnKeyDown Property

Occurs when a user presses any key while the control has focus.

**property** OnKeyDown;

#### Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys, and pressed mouse buttons.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

### 1.19.1.2.2.57 TDesignerObjTree.OnKeyPress Property

Occurs when key pressed.

**property** OnKeyPress;

#### Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

### 1.19.1.2.2.58 TDesignerObjTree.OnKeyUp Property

Occurs when the user releases a key that has been pressed.

**property** OnKeyUp;

#### Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

### 1.19.1.2.2.59 TDesignerObjTree.OnMouseDown Property

Occurs when the user presses a mouse button with the mouse pointer over a control.

**property** OnMouseDown;

#### Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

### 1.19.1.2.2.60 TDesignerObjTree.OnMouseMove Property

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

**property** OnMouseMove;

#### Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

### 1.19.1.2.2.61 TDesignerObjTree.OnMouseUp Property

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

**property** OnMouseUp;

#### Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

### 1.19.1.2.2.62 TDesignerObjTree.OnStartDock Property

Occurs when the user begins to drag a control with a DragKind of dkDock.

**property** OnStartDock;

#### Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

### 1.19.1.2.2.63 TDesignerObjTree.OnStartDrag Property

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

**property** OnStartDrag;

#### Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itse

### 1.19.1.2.2.64 TDesignerObjTree.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

**property** ParentBiDiMode;

### 1.19.1.2.2.65 TDesignerObjTree.ParentColor Property

Determines where a control looks for its color information.

**property** ParentColor;

### 1.19.1.2.2.66 TDesignerObjTree.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

**property** ParentCtl3D;

#### Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

### 1.19.1.2.2.67 TDesignerObjTree.ParentFont Property

Determines where a control looks for its font information.

**property** ParentFont;

### 1.19.1.2.2.68 TDesignerObjTree.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

**property** ParentShowHint;



### 1.19.1.2.2.69 TDesignerObjTree.PopupMenu Property

Identifies the pop-up menu associated with the control.

```
property PopupMenu;
```

#### Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

### 1.19.1.2.2.70 TDesignerObjTree.ReadOnly Property

Determines whether the user can edit the node labels.

```
property ReadOnly;
```

#### Description

Use ReadOnly to specify whether the user can edit the nodes of the tree view. If ReadOnly is true, the user can expand and collapse nodes, but can't edit their labels. If ReadOnly is false, the user can edit the labels as well. The default value is false.

### 1.19.1.2.2.71 TDesignerObjTree.RightClickSelect Property

Determines whether the Selected property returns nodes that are selected using the right mouse button.

```
property RightClickSelect;
```

#### Description

Use RightClickSelect to allow the Selected property to indicate nodes the user clicks with the right mouse button. If RightClickSelect is true, the value of Selected is the value of the node last clicked with either the right or left mouse button. If RightClickSelect is false, the value of Selected is the node last clicked using the left mouse button.

RightClickSelect affects only the value of the Selected property. It does not cause the tree view to highlight a new node if the node is selected using the right mouse button.

#### Notes

RightClickSelect must be set to true before the user right-clicks the tree view for it to affect the value of the Selected property.

### 1.19.1.2.2.72 TDesignerObjTree.RowSelect Property

Specifies whether the entire row of the selected item is highlighted.

```
property RowSelect;
```

#### Description

Set RowSelect to true to cause the entire row of the selected item to be highlighted.

RowSelect is ignored if ShowLines is true.

### 1.19.1.2.2.73 TDesignerObjTree.ShowButtons Property

Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item.

```
property ShowButtons;
```

#### Description

If ShowButtons is true, a button will appear to the left of each parent item. The user can click the button to expand or collapse the child items as an alternative to double-clicking the parent item.

#### 1.19.1.2.2.74 TDesignerObjTree.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

#### 1.19.1.2.2.75 TDesignerObjTree.ShowLines Property

Specifies whether to display the lines that link child nodes to their corresponding parent nodes.

```
property ShowLines;
```

##### Description

If ShowLines is true, lines linking child nodes to their parent nodes are displayed. Nodes at the root of the hierarchy are not automatically linked. To link nodes at the root, the ShowRoot property must also be set to true.

#### 1.19.1.2.2.76 TDesignerObjTree.ShowRoot Property

Specifies whether lines connecting top-level nodes are displayed.

```
property ShowRoot;
```

##### Description

To show lines connecting top-level nodes to a single root, set the tree view's ShowRoot and ShowLines properties to true.

#### 1.19.1.2.2.77 TDesignerObjTree.SortType Property

Determines if and how the nodes in a tree view are automatically sorted.

```
property SortType;
```

##### Description

Once a tree view is sorted, the original hierarchy is lost. That is, setting the SortType back to stNone will not restore the original order of items. These are the possible values:

#### 1.19.1.2.2.78 TDesignerObjTree.StateImages Property

Determines which image list to use for state images.

```
property StateImages;
```

##### Description

Use StateImages to provide a set of bitmaps that reflect the state of tree view nodes. The state image appears as an additional image to the left of the item's icon.

#### 1.19.1.2.2.79 TDesignerObjTree.TabOrder Property

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

#### 1.19.1.2.2.80 TDesignerObjTree.TabStop Property

Determines if the user can tab to a control.

```
property TabStop;
```

##### Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

1.19.1.2.2.81 TDesignerObjTree.ToolTips Property

Specifies whether the items in the tree view have tool tips.

**property** ToolTips;

Description

Set ToolTips to true to specify that items in the tree view control have tool tips (Help Hints).

Specify the ToolTip text in an OnHint event handler using the Hint property.

1.19.1.2.2.82 TDesignerObjTree.Visible Property

Determines whether the component appears on screen.

**property** Visible;

Description



Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

1.19.2 Functions

The following table lists functions in this documentation.

Functions

Function	Description
 CreateGhosedImages (  see page 294)	Creates ghosed images and adds them to the image list.

Legend

	Method
-------------------------------------------------------------------------------------	--------

1.19.2.1 ed\_ObjTree.CreateGhosedImages Function

Creates ghosed images and adds them to the image list.

**procedure** CreateGhosedImages(IL: TCustomImageList);



File

ed\_ObjTree

1.19.3 Types

The following table lists types in this documentation.

Types

Type	Description
TCreateSprigNodeEvent (  see page 295)	See TCustomDesignerObjTree.OnCreateSprigNode Event (  see page 271)

### 1.19.3.1 ed\_ObjTree.TCreateSprigNodeEvent Type

See TCustomDesignerObjTree.OnCreateSprigNode Event ( see page 271)

```
TCreateSprigNodeEvent = procedure (Sender: TObject; Sprig: TSprig; var Text: string; var
Accept: Boolean) of object;
```

**File**

ed\_ObjTree

## 1.20 ed\_TextEdit Namespace

### 1.20.1 Classes

The following table lists classes in this documentation.

**Classes**

Class	Description
TDsnInplaceEditor ( see page 295)	Design in-place editor control.
TInplaceComponentEditor ( see page 302)	Designer adapter for in-place text editing.

#### 1.20.1.1 TDsnInplaceEditor Class

Design in-place editor control.

**Class Hierarchy**



```
TDsnInplaceEditor = class(TCustomEditEx);
```

**File**

ed\_TextEdit

**Description**

This control is created and owned by designer (instance of TzFormDesigner). Properties and position of the in-place editor are defined by using in-place adapters, objects derived from TInplaceComponentEditor ( see page 302). In-place edit control can not be accessed via designer, it is only passed in TInplaceComponentEditor.SetEditor ( see page 304) method to adjust editor properties.


















**Members**

**TUnicodeEdit Methods**














TUnicodeEdit Methods	Description
Create ( see page 108)	Creates and initializes a TUnicodeEdit instance.
Destroy ( see page 108)	Destroys an instance of TUnicodeEdit.

**TBtnEdit Class**






TBtnEdit Class	Description
AdjustClientRect ( see page 73)	Overrides the inherited method.
ButtonClick ( see page 73)	Simulates a button click, as if the user had clicked the button.

 <b>Create</b> ( <a href="#">see page 73</a> )	Creates and initializes a TBtnEdit ( <a href="#">see page 70</a> ) instance.
 <b>CreateParams</b> ( <a href="#">see page 73</a> )	Overrides the inherited method.
 <b>CreateWnd</b> ( <a href="#">see page 73</a> )	Overrides the inherited method.
 <b>Destroy</b> ( <a href="#">see page 74</a> )	Destroys an instance of TBtnEdit ( <a href="#">see page 70</a> )
 <b>EndTracking</b> ( <a href="#">see page 74</a> )	Called after finishing the mouse tracking
 <b>KeyDown</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
 <b>KeyPress</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
 <b>MouseMove</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
 <b>MouseUp</b> ( <a href="#">see page 75</a> )	Overrides the inherited method.
 <b>Paint</b> ( <a href="#">see page 75</a> )	Overrides the base rendering method.
 <b>PaintBtnGlyph</b> ( <a href="#">see page 75</a> )	Renders the image of the button.
 <b>PaintStatus</b> ( <a href="#">see page 75</a> )	Paints status area. This method is called only if StatusWidth ( <a href="#">see page 77</a> ) is greater 0.
 <b>PaintWindow</b> ( <a href="#">see page 75</a> )	Overrides the inherited method.
 <b>PtInButton</b> ( <a href="#">see page 75</a> )	Checks if specified point is over the button
 <b>StartTracking</b> ( <a href="#">see page 76</a> )	Calls immediately after user pushes the button.
 <b>StopTracking</b> ( <a href="#">see page 76</a> )	Calls immediately after user releases the button.
 <b>TrackButton</b> ( <a href="#">see page 76</a> )	Controls tracking button process.





### TCustomEditEx Class

<b>TCustomEditEx Class</b>	<b>Description</b>
 <b>AcceptListValue</b> ( <a href="#">see page 82</a> )	Active pop-up window.
 <b>ButtonClick</b> ( <a href="#">see page 82</a> )	Simulates a button click, as if the user had clicked the button.
 <b>CloseUp</b> ( <a href="#">see page 82</a> )	Generates an OnCloseUp ( <a href="#">see page 85</a> ) event and makes some other actions.
 <b>Create</b> ( <a href="#">see page 82</a> )	Creates and initializes a TBtnEdit instance.
 <b>Destroy</b> ( <a href="#">see page 82</a> )	Destroys an instance of TBtnEdit
 <b>DoDropDownKeys</b> ( <a href="#">see page 83</a> )	Works as a method dispatcher depending on parameter values.
 <b>DropDown</b> ( <a href="#">see page 83</a> )	Generates an OnDropDown ( <a href="#">see page 85</a> ) event.
 <b>EndTracking</b> ( <a href="#">see page 83</a> )	Called after finishing the mouse tracking
 <b>KeyPress</b> ( <a href="#">see page 83</a> )	Respond to keyboard input.
 <b>MouseDown</b> ( <a href="#">see page 83</a> )	Overrides inherited method
 <b>MouseMove</b> ( <a href="#">see page 83</a> )	Overrides the inherited method.
 <b>PaintBtnGlyph</b> ( <a href="#">see page 84</a> )	Overrides inherited method
 <b>StartTracking</b> ( <a href="#">see page 84</a> )	Overrides inherited property







### TDsnInplaceEditor Class




<b>TDsnInplaceEditor Class</b>	<b>Description</b>
 <b>Close</b> ( <a href="#">see page 299</a> )	Closes in-place text editor.
 <b>Create</b> ( <a href="#">see page 299</a> )	Creates and initializes a TBtnEdit instance.
 <b>Destroy</b> ( <a href="#">see page 300</a> )	Destroys an instance of TDsnInplaceEditor.
 <b>LoadText</b> ( <a href="#">see page 300</a> )	Loads text from adapter to edit control.
 <b>SaveText</b> ( <a href="#">see page 300</a> )	Saves text from edit control to adapter.

### TUnicodeEdit Properties





<b>TUnicodeEdit Properties</b>	<b>Description</b>
 <b>IsUnicode</b> ( <a href="#">see page 108</a> )	Specifies whether control is Unicode edit.
 <b>SelTextW</b> ( <a href="#">see page 109</a> )	Specifies the selected portion of the edit control's text (Unicode version).
 <b>Text</b> ( <a href="#">see page 109</a> )	Specifies the text string that is displayed in the edit box (Ansi version).
 <b>TextW</b> ( <a href="#">see page 109</a> )	Specifies the text string that is displayed in the edit box (Ansi version).

### TBtnEdit Class










<b>TBtnEdit Class</b>	<b>Description</b>
 <b>Alignment</b> ( <a href="#">see page 76</a> )	Determines how the text is aligned within the editor control.
 <b>ButtonVisible</b> ( <a href="#">see page 76</a> )	Specifies if button-like rectangle at the right edge of the control is visible.
 <b>ButtonWidth</b> ( <a href="#">see page 77</a> )	Specifies width in pixels of the button.
 <b>Canvas</b> ( <a href="#">see page 77</a> )	Provides access to the drawing surface of the TBtnEdit ( <a href="#">see page 70</a> ).
 <b>MultiLine</b> ( <a href="#">see page 77</a> )	Designates a multiline edit control. The default is single-line edit control.
 <b>StatusWidth</b> ( <a href="#">see page 77</a> )	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.

 WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
 WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

**TCustomEditEx Class**

TCustomEditEx Class	Description
 ActiveList (see page 84)	Specifies current popup control.
 EditStyle (see page 84)	Specifies edit style of the button
 ListAlign (see page 84)	Specifies relative align of popup control.
 PickList (see page 84)	Determines the drop-down list.





**TDsnInplaceEditor Class**

TDsnInplaceEditor Class	Description
 Adapter (see page 300)	Specifies associated in-place adapter.
 Alignment (see page 300)	Determines how the text is aligned within the editor control.
 Color (see page 300)	Specifies the background color of the control.
 IsUnicode (see page 301)	Specifies whether control is Unicode edit.
 MultiLine (see page 301)	Designates a multiline edit control. The default is single-line edit control.
 OnChange (see page 301)	Occurs when the text for the edit control may have changed.
 OnExit (see page 301)	Occurs when the input focus shifts away from one control to another.
 TextW (see page 301)	Specifies the text string that is displayed in the edit box (Ansi version).
 WordWrap (see page 302)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.







**TBtnEdit Events****TBtnEdit Class**

TBtnEdit Class	Description
 OnButtonClick (see page 78)	Occurs when the user clicks the button.

**TCustomEditEx Class**

TCustomEditEx Class	Description
 OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
 OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
 OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
 OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.





**Legend**

	Constructor
	virtual
	protected
	Property
	read only
	Event

**TBtnEdit Events****TBtnEdit Class**

TBtnEdit Class	Description
 OnButtonClick (see page 78)	Occurs when the user clicks the button.







































**TCustomEditEx Class**

TCustomEditEx Class	Description
 OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
 OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
 OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
 OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.



























**TUnicodeEdit Methods**

TUnicodeEdit Methods	Description
  Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
  Destroy (see page 108)	Destroys an instance of TUnicodeEdit.









**TBtnEdit Class**

TBtnEdit Class	Description
  AdjustClientRect (see page 73)	Overrides the inherited method.
  ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
  Create (see page 73)	Creates and initializes a TBtnEdit (see page 70) instance.
  CreateParams (see page 73)	Overrides the inherited method.
  CreateWnd (see page 73)	Overrides the inherited method.
  Destroy (see page 74)	Destroys an instance of TBtnEdit (see page 70)
  EndTracking (see page 74)	Called after finishing the mouse tracking
  KeyDown (see page 74)	Overrides the inherited method.
  KeyPress (see page 74)	Overrides the inherited method.
  MouseMove (see page 74)	Overrides the inherited method.
  MouseUp (see page 75)	Overrides the inherited method.
  Paint (see page 75)	Overrides the base rendering method.
  PaintBtnGlyph (see page 75)	Renders the image of the button.
  PaintStatus (see page 75)	Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.
  PaintWindow (see page 75)	Overrides the inherited method.
  PtInButton (see page 75)	Checks if specified point is over the button
  StartTracking (see page 76)	Calls immediately after user pushes the button.
  StopTracking (see page 76)	Calls immediately after user releases the button.
  TrackButton (see page 76)	Controls tracking button process.





**TCustomEditEx Class**

TCustomEditEx Class	Description
  AcceptListValue (see page 82)	Active pop-up window.
  ButtonClick (see page 82)	Simulates a button click, as if the user had clicked the button.
  CloseUp (see page 82)	Generates an OnCloseUp (see page 85) event and makes some other actions.
  Create (see page 82)	Creates and initializes a TBtnEdit instance.
  Destroy (see page 82)	Destroys an instance of TBtnEdit
  DoDropDownKeys (see page 83)	Works as a method dispatcher depending on parameter values.
  DropDown (see page 83)	Generates an OnDropDown (see page 85) event.
  EndTracking (see page 83)	Called after finishing the mouse tracking
  KeyPress (see page 83)	Respond to keyboard input.
  MouseDown (see page 83)	Overrides inherited method
  MouseMove (see page 83)	Overrides the inherited method.
  PaintBtnGlyph (see page 84)	Overrides inherited method
  StartTracking (see page 84)	Overrides inherited property










**TDsnInplaceEditor Class**

TDsnInplaceEditor Class	Description
  Close (see page 299)	Closes in-place text editor.
  Create (see page 299)	Creates and initializes a TBtnEdit instance.
  Destroy (see page 300)	Destroys an instance of TDsnInplaceEditor.
  LoadText (see page 300)	Loads text from adapter to edit control.
  SaveText (see page 300)	Saves text from edit control to adapter.





**TUnicodeEdit Properties**

TUnicodeEdit Properties	Description
 IsUnicode (see page 108)	Specifies whether control is Unicode edit.
 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).










**TBtnEdit Class**

TBtnEdit Class	Description
 Alignment (see page 76)	Determines how the text is aligned within the editor control.
 ButtonVisible (see page 76)	Specifies if button-like rectangle at the right edge of the control is visible.
 ButtonWidth (see page 77)	Specifies width in pixels of the button.
 Canvas (see page 77)	Provides access to the drawing surface of the TBtnEdit (see page 70).
 MultiLine (see page 77)	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth (see page 77)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
 WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

**TCustomEditEx Class**

TCustomEditEx Class	Description
 ActiveList (see page 84)	Specifies current popup control.
 EditStyle (see page 84)	Specifies edit style of the button
 ListAlign (see page 84)	Specifies relative align of popup control.
 PickList (see page 84)	Determines the drop-down list.

**TDsnInplaceEditor Class**

TDsnInplaceEditor Class	Description
 Adapter (see page 300)	Specifies associated in-place adapter.
 Alignment (see page 300)	Determines how the text is aligned within the editor control.
 Color (see page 300)	Specifies the background color of the control.
 IsUnicode (see page 301)	Specifies whether control is Unicode edit.
 MultiLine (see page 301)	Designates a multiline edit control. The default is single-line edit control.
 OnChange (see page 301)	Occurs when the text for the edit control may have changed.
 OnExit (see page 301)	Occurs when the input focus shifts away from one control to another.
 TextW (see page 301)	Specifies the text string that is displayed in the edit box (Ansi version).
 WordWrap (see page 302)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

**1.20.1.1.1 TDsnInplaceEditor Methods****1.20.1.1.1.1 TDsnInplaceEditor.Close Method**

Closes in-place text editor.

```
procedure Close(Accept: Boolean);
```

**Description**

If Accept is True edited text will be saved to control otherwise all not saved changes will be canceled.

**1.20.1.1.1.2 TDsnInplaceEditor.Create Constructor**

Creates and initializes a TBtnEdit instance.

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

**Description**

Use Create to programmatically instantiate this type of a control.

**Create**

- Calls the inherited Create method
- Sets the width of the button calling GetSystemMetrics method with SM\_CXVSCROLL parameter
- Sets ButtonVisible to false



- Sets Alignment to taLeftJustify
- Sets MultiLine to true
- Creates Canvas object and sets its Control property to the control itself.

#### 1.20.1.1.1.3 TDsnInplaceEditor.Destroy Destructor

Destroys an instance of TDsnInplaceEditor.

```
destructor Destroy; override;
```

##### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

#### 1.20.1.1.1.4 TDsnInplaceEditor.LoadText Method

Loads text from adapter to edit control.

```
procedure LoadText;
```

#### 1.20.1.1.1.5 TDsnInplaceEditor.SaveText Method

Saves text from edit control to adapter.

```
procedure SaveText;
```

### 1.20.1.1.2 TDsnInplaceEditor Properties

#### 1.20.1.1.2.1 TDsnInplaceEditor.Adapter Property

Specifies associated in-place adapter.

```
property Adapter: TInplaceComponentEditor;
```

##### Description

Assigning new adapter will destroy (see page 300) previous adapter. Assigned adapter is managed by in-place editor, so it has not be freed outside editor.

#### 1.20.1.1.2.2 TDsnInplaceEditor.Alignment Property

Determines how the text is aligned within the editor control.

```
property Alignment: TAlignment;
```

##### Description

Use Alignment to change the way the text is formatted by the in-place editor control. Alignment can take one of the following values:

Value	Meaning
taLeftJustify	Align text to the left side of the control
taCenter	Center text horizontally in the control
taRightJustify	Align text to the right side of the control

#### 1.20.1.1.2.3 TDsnInplaceEditor.Color Property

Specifies the background color of the control.

```
property Color;
```

**Description**

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

**1.20.1.1.2.4 TDsnInplaceEditor.IsUnicode Property**

Specifies whether control is Unicode edit.

```
property IsUnicode: Boolean;
```

**Description**

Set IsUnicode to True to make edit control Unicode window. When control is Unicode TextW (see page 109) and SelTextW (see page 109) properties should be used instead of Text and SelText properties.

**1.20.1.1.2.5 TDsnInplaceEditor.MultiLine Property**

Designates a multiline edit control. The default is single-line edit control.

```
property MultiLine: Boolean;
```

**Description**

When MultiLine is True TBtnEdit is equivalent to TMemo control, otherwise it is equivalent to TEdit control.

**1.20.1.1.2.6 TDsnInplaceEditor.OnChange Property**

Occurs when the text for the edit control may have changed.

```
property OnChange;
```

**Description**

Write an OnChange event handler to take specific action whenever the text for the edit control may have changed. Use the Modified property to see if a change actually occurred. The Text property of the edit control will already be updated to reflect any changes. This event provides the first opportunity to respond to modifications that the user types into the edit control.

**1.20.1.1.2.7 TDsnInplaceEditor.OnExit Property**

Occurs when the input focus shifts away from one control to another.

```
property OnExit;
```

**Description**

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

**1.20.1.1.2.8 TDsnInplaceEditor.TextW Property**

Specifies the text string that is displayed in the edit box (Ansi version).

```
property TextW: WideString;
```

**Description**

Use the TextW property to read the text of the edit box or specify a new string for the TextW value. By default, TextW is the string specified in the Name property.

Use this property when IsUnicode is True. Otherwise this property may corrupt string due to Ansi to Unicode conversion.

### 1.20.1.1.2.9 TDsnInplaceEditor.WordWrap Property

Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

**property** WordWrap: Boolean;

#### Description

Set WordWrap to true to make the edit control wrap text at the right margin so it fits in the client area. The wrapping is cosmetic only. The text does not include any return characters that were not explicitly entered. Set WordWrap to false to have the edit control show a separate line only where return characters were explicitly entered into the text.

#### Notes

There should be no use for a horizontal scroll bar if WordWrap is true.

## 1.20.1.2 TInplaceComponentEditor Class

Designer adapter for in-place text editing.

#### Class Hierarchy

ed\_TextEdit.TInplaceComponentEditor

TInplaceComponentEditor = **class**;

#### File

ed\_TextEdit

#### Description












TInplaceComponentEditor objects are created by designer to handle in-place text editing. Classes derived from TInplaceComponentEditor defines editable text, rectangle of in-place editor and its properties.

One control may have several text regions, handling of this regions are provided by classes derived from this class.





After implementing new in-place editor adapter you have to register it using RegisterInplaceComponentEditor (see page 306).

#### Members






##### TInplaceComponentEditor Methods

TInplaceComponentEditor Methods	Description
 Create (see page 303)	Creates and initializes a TInplaceComponentEditor instance.
 GetBoundRect (see page 303)	Returns bounding rectangle for in-place editor.
 GetText (see page 304)	Returns control text to be edited.
 GetTextW (see page 304)	Returns control Unicode text to be edited.
 HandlePos (see page 304)	Specifies whether control text may be edited at specified position.
 IsAutoUpdate (see page 304)	Specifies whether text has to be written on any change in in-place editor.
 IsUnicode (see page 304)	Determines whether edited text is Unicode.
 SetEditor (see page 304)	Adjusts in-place editor properties.
 SetHitPoint (see page 305)	Called before activating in-place editor to define editing region.
 SetText (see page 305)	Writes edited text to control.
 SetTextW (see page 305)	Writes edited Unicode text to control.























##### TInplaceComponentEditor Properties

TInplaceComponentEditor Properties	Description
 BoundRect (see page 305)	Returns bounding rectangle for in-place editor.
 Control (see page 305)	Control managed by in-place adapter.
 Text (see page 305)	Returns control text to be edited.
 TextW (see page 306)	Returns control Unicode text to be edited.







**Legend**

	Constructor
	virtual
	protected
	Property
	read only

**TInplaceComponentEditor Methods**

TInplaceComponentEditor Methods	Description
  Create (see page 303)	Creates and initializes a TInplaceComponentEditor instance.
  GetBoundRect (see page 303)	Returns bounding rectangle for in-place editor.
  GetText (see page 304)	Returns control text to be edited.
  GetTextW (see page 304)	Returns control Unicode text to be edited.
  HandlePos (see page 304)	Specifies whether control text may be edited at specified position.
  IsAutoUpdate (see page 304)	Specifies whether text has to be written on any change in in-place editor.
  IsUnicode (see page 304)	Determines whether edited text is Unicode.
  SetEditor (see page 304)	Adjusts in-place editor properties.
  SetHitPoint (see page 305)	Called before activating in-place editor to define editing region.
  SetText (see page 305)	Writes edited text to control.
  SetTextW (see page 305)	Writes edited Unicode text to control.

**TInplaceComponentEditor Properties**

TInplaceComponentEditor Properties	Description
  BoundRect (see page 305)	Returns bounding rectangle for in-place editor.
  Control (see page 305)	Control managed by in-place adapter.
 Text (see page 305)	Returns control text to be edited.
 TextW (see page 306)	Returns control Unicode text to be edited.

**1.20.1.2.1 TInplaceComponentEditor Methods****1.20.1.2.1.1 TInplaceComponentEditor.Create Constructor**

Creates and initializes a TInplaceComponentEditor instance.

```
constructor Create(AControl: TControl); virtual;
```

**Description**

Use Create to programmatically instantiate a TInplaceComponentEditor object.

TInplaceComponentEditor objects are created by designer to handle in-place text editing. Classes derived from TInplaceComponentEditor defines editable text, rectangle of in-place editor and its properties.

One control may have several text regions, handling of this regions are provided by classes derived from this class.

After implementing new in-place editor adapter you have to register it using RegisterInplaceComponentEditor (see page 306).

**1.20.1.2.1.2 TInplaceComponentEditor.GetBoundRect Method**

Returns bounding rectangle for in-place editor.

```
function GetBoundRect: TRect; virtual;
```

**Description**

Implement this method in derived classes to calculate position of in-place editor. If control has several editable regions define particular region in SetHitPoint (see page 305) method and use it in GetBoundRect to return rectangle of the region.

### 1.20.1.2.1.3 TInplaceComponentEditor.GetText Method

Returns control text to be edited.

```
function GetText: string; virtual;
```

#### Description

By default, GetText returns value of TControl.Caption. Implement this method in derived classes to return text associated with particular text region defined by the SetHitPoint (see page 305) method.

### 1.20.1.2.1.4 TInplaceComponentEditor.GetTextW Method

Returns control Unicode text to be edited.

```
function GetTextW: WideString; virtual;
```

#### Description

By default, GetTextW returns value of TControl.Caption. Implement this method in derived classes to return text associated with particular text region defined by the SetHitPoint (see page 305) method.

### 1.20.1.2.1.5 TInplaceComponentEditor.HandlePos Method

Specifies whether control text may be edited at specified position.

```
function HandlePos(Pos: TPoint): Boolean; virtual;
```

#### Description

Write this method to define ability to activate in-place editor at specified position. If Pos = Point(-1, -1), in-place editor is activated by keyboard (user presses Enter key).

### 1.20.1.2.1.6 TInplaceComponentEditor.IsAutoUpdate Method

Specifies whether text has to be written on any change in in-place editor.

```
function IsAutoUpdate: Boolean; virtual;
```

#### Description

If IsAutoUpdate returns True, any text input in in-place editor immediately saved to control, in-place editor bounds are recalculated.

Otherwise text is written to control when editor loses focus or when user presses Enter key in editor.

By default, IsAutoUpdate returns False.

### 1.20.1.2.1.7 TInplaceComponentEditor.IsUnicode Method

Determines whether edited text is Unicode.

```
function IsUnicode: Boolean; virtual;
```

### 1.20.1.2.1.8 TInplaceComponentEditor.SetEditor Method

Adjusts in-place editor properties.

```
procedure SetEditor(Editor: TDsnInplaceEditor); virtual;
```

#### Description

Set properties of in-place editor control (TDsnInplaceEditor (see page 295)). Override this method to define particular properties of in-place editor for current editing region.

Default settings:

```
procedure TInplaceComponentEditor.SetEditor(Editor: TDsnInplaceEditor);  
begin
```

```
// Default settings
Editor.Color := TControlAccess(Control).Color;
Editor.Font := TControlAccess(Control).Font;
Editor.MultiLine := False;
Editor.WordWrap := False;
Editor.Alignment := taLeftJustify;
Editor.WantTabs := False;
end;
```

#### 1.20.1.2.1.9 TInplaceComponentEditor.SetHitPoint Method

Called before activating in-place editor to define editing region.

```
procedure SetHitPoint(Pos: TPoint); virtual;
```

##### Description

Control (see page 305) may have many editable regions, for example, in list view column headers, item captions and sub-items may be edited. This method allows to define edited region, save it in adapter and use in all other adapter methods.

#### 1.20.1.2.1.10 TInplaceComponentEditor.SetText Method

Writes edited text to control.

```
procedure SetText(const Value: string); virtual;
```

##### Description

By default, SetText writes value to TControl.Caption property. Override this method to write edited text to another destination.

#### 1.20.1.2.1.11 TInplaceComponentEditor.SetTextW Method

Writes edited Unicode text to control.

```
procedure SetTextW(const Value: WideString); virtual;
```

##### Description

By default, SetTextW writes value to TControl.Caption property. Override this method to write edited text to another destination.

### 1.20.1.2.2 TInplaceComponentEditor Properties

#### 1.20.1.2.2.1 TInplaceComponentEditor.BoundsRect Property

Returns bounding rectangle for in-place editor.

```
property BoundsRect: TRect;
```

##### Description

BoundsRect returns rectangle of particular editable region defined in SetHitPoint (see page 305) method.

#### 1.20.1.2.2.2 TInplaceComponentEditor.Control Property

Control managed by in-place adapter.

```
property Control: TControl;
```

##### Description

Reference to Control is passed in constructor.

#### 1.20.1.2.2.3 TInplaceComponentEditor.Text Property

Returns control text to be edited.

**property** Text: **string**;

**Description**

By default, Text is associated with TControl.Caption property. Get and set methods of this property may be overridden in derived classes to access other control texts.

1.20.1.2.2.4 **TInplaceComponentEditor.TextW Property**

Returns control Unicode text to be edited.

**property** TextW: **WideString**;

**Description**


By default, TextW is associated with TControl.Caption property. Get and set methods of this property may be overridden in derived classes to access other control texts.

---

## 1.20.2 Functions

The following table lists functions in this documentation.

**Functions**

Function	Description
 CreateImplEditor (  see page 306)	Creates in-place text adapter for specified Control.
 RegisterInplaceComponentEditor (  see page 306)	Registers in-place text adapter.

**Legend**

	Method
-------------------------------------------------------------------------------------	--------

### 1.20.2.1 ed\_TextEdit.CreateImplEditor Function

**function** CreateImplEditor(Control: TControl): TInplaceComponentEditor;

**File**

ed\_TextEdit

**Description**

Creates in-place text adapter for specified Control.

### 1.20.2.2 ed\_TextEdit.RegisterInplaceComponentEditor Function

Registers in-place text adapter.

**procedure** RegisterInplaceComponentEditor(ControlClass: TControlClass; EditorClass: TInplaceComponentEditorClass);

**File**

ed\_TextEdit

**Description**

Associates in-place text adapter EditorClass with the control class.

To disable in-place editing pass EditorClass = nil.

## 1.20.3 Types

The following table lists types in this documentation.

### Types

Type	Description
TInplaceComponentEditorClass ( <a href="#">see page 307</a> )	TInplaceComponentEditor ( <a href="#">see page 302</a> ) class reference.

### 1.20.3.1 ed\_TextEdit.TInplaceComponentEditorClass Type

```
TInplaceComponentEditorClass = class of TInplaceComponentEditor;
```

#### File

ed\_TextEdit

#### Description

TInplaceComponentEditor ([see page 302](#)) class reference.

## 1.21 edActns Namespace

### 1.21.1 Classes

The following table lists classes in this documentation.

#### Classes

Class	Description
TDesignerAction ( <a href="#">see page 308</a> )	TDesignerAction is the base class for design actions meant to be used with active designer
TdsnAlignmentDlg ( <a href="#">see page 311</a> )	Opens the Alignment dialog box for the current designer
TdsnAlignToGrid ( <a href="#">see page 313</a> )	Performs Align To Grid action for the active designer
TdsnBringToFront ( <a href="#">see page 315</a> )	Performs Bring To Front action for the active designer
TdsnCopy ( <a href="#">see page 316</a> )	Performs Copy to clipboard action for the active designer.
TdsnCreationOrderDlg ( <a href="#">see page 317</a> )	Opens the Creation Order dialog box for the active designer.
TdsnCut ( <a href="#">see page 318</a> )	Performs Cut To Clipboard action for the active designer.
TdsnDelete ( <a href="#">see page 318</a> )	Performs Delete action for the current designer
TdsnDesignMode ( <a href="#">see page 319</a> )	Performs switching design mode for the current form with the active designer
TdsnFlipChildren ( <a href="#">see page 320</a> )	Performs Flip Children action for the selected controls.
TdsnFlipChildrenAll ( <a href="#">see page 322</a> )	Performs Flip Children action for all controls.
TdsnGroupControls ( <a href="#">see page 323</a> )	Group Controls action for the active designer.
TdsnLockControls ( <a href="#">see page 325</a> )	Performs "Lock Controls" action for the active designer
TdsnPaste ( <a href="#">see page 327</a> )	Perform Paste action for the current designer
TdsnRedo ( <a href="#">see page 327</a> )	Repeat last undone operation.
TdsnScale ( <a href="#">see page 327</a> )	Opens the Scale dialog box for the active designer
TDsnSelAction ( <a href="#">see page 329</a> )	Base class of designer actions that require not empty selection
TdsnSelectAll ( <a href="#">see page 330</a> )	Performs Select All action for the active designer
TdsnSendToBack ( <a href="#">see page 330</a> )	Performs Send To Back action for the active designer
TdsnShowTabOrder ( <a href="#">see page 332</a> )	Set show tab order design mode.
TdsnSizeDlg ( <a href="#">see page 334</a> )	Opens the Size dialog box.



TdsnTabOrderDlg (🔗 see page 335)	Opens the Edit Tab Order dialog box
TdsnTargetAction (🔗 see page 337)	Designer target actions. Determines whether target is managed by designer.
TdsnTextEditMode (🔗 see page 337)	Toggles TextEditMode of the active designer.
TdsnUndo (🔗 see page 338)	Backs out last change in the undo buffer.
TdsnUngroupControls (🔗 see page 339)	Ungroups selected controls.

## 1.21.1.1 TDesignerAction Class

TDesignerAction is the base class for design actions meant to be used with active designer

### Class Hierarchy



```
TDesignerAction = class(TCustomAction);
```

### File

edActns

### Description

TDesignerAction introduces support for the for the specific behaviour of designer actions.

It publishes some properties of the TCustomAction to use in its descendants and overrides Update (🔗 see page 309) method to interact with designer's Active property (disabling when there is no active designer).

Use it as a base class when deriving your own designer actions.

### Members

#### TDesignerAction Methods

TDesignerAction Methods	Description
🔗 Update (🔗 see page 309)	Indicates whether the action updates itself.

#### TDesignerAction Properties

TDesignerAction Properties	Description
🔗 Caption (🔗 see page 309)	Represents the caption of client controls and menu items.
🔗 Enabled (🔗 see page 309)	Indicates whether client controls and menu items are enabled.
🔗 HelpContext (🔗 see page 309)	Indicates the help context ID for client controls and menu items.
🔗 HelpKeyword (🔗 see page 310)	Indicates the help keyword for client controls and menu items.
🔗 HelpType (🔗 see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
🔗 Hint (🔗 see page 310)	Indicates the Help hint for client controls and menu items.
🔗 ImageIndex (🔗 see page 310)	Indicates the ImageIndex property for client controls and menu items.
🔗 OnExecute (🔗 see page 310)	Occurs when the client event that is linked to it fires.
🔗 OnHint (🔗 see page 310)	Occurs when the mouse pauses over a client control or menu item.
🔗 OnUpdate (🔗 see page 311)	Occurs when the application is idle or when the action list updates.
🔗 SecondaryShortCuts (🔗 see page 311)	Specifies the short cuts (in addition to ShortCut (🔗 see page 311)) for triggering clients.
🔗 ShortCut (🔗 see page 311)	Specifies the ShortCut property for client menu items.
🔗 Visible (🔗 see page 311)	Specifies the Visible property for client controls and menu items.














### Legend

🔗	Method
🔗	virtual
🔗	Property

#### TDesignerAction Methods

TDesignerAction Methods	Description
🔗 Update (🔗 see page 309)	Indicates whether the action updates itself.

**TDesignerAction Properties**

<b>TDesignerAction Properties</b>	<b>Description</b>
 Caption ( <a href="#">see page 309</a> )	Represents the caption of client controls and menu items.
 Enabled ( <a href="#">see page 309</a> )	Indicates whether client controls and menu items are enabled.
 HelpContext ( <a href="#">see page 309</a> )	Indicates the help context ID for client controls and menu items.
 HelpKeyword ( <a href="#">see page 310</a> )	Indicates the help keyword for client controls and menu items.
 HelpType ( <a href="#">see page 310</a> )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint ( <a href="#">see page 310</a> )	Indicates the Help hint for client controls and menu items.
 ImageIndex ( <a href="#">see page 310</a> )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute ( <a href="#">see page 310</a> )	Occurs when the client event that is linked to it fires.
 OnHint ( <a href="#">see page 310</a> )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate ( <a href="#">see page 311</a> )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts ( <a href="#">see page 311</a> )	Specifies the short cuts (in addition to ShortCut ( <a href="#">see page 311</a> )) for triggering clients.
 ShortCut ( <a href="#">see page 311</a> )	Specifies the ShortCut property for client menu items.
 Visible ( <a href="#">see page 311</a> )	Specifies the Visible property for client controls and menu items.

**1.21.1.1.1 TDesignerAction Methods****1.21.1.1.1.1 TDesignerAction.Update Method**

Indicates whether the action updates itself.

```
function Update: Boolean; override;
```

**Description**

The Update method sets Enable property and Result to True if there is active designer in DsnManager ([see page 514](#)).

**1.21.1.1.2 TDesignerAction Properties****1.21.1.1.2.1 TDesignerAction.Caption Property**

Represents the caption of client controls and menu items.

```
property Caption;
```

**Description**

This is inherited property.

See TCustomAction.Caption for details.

**1.21.1.1.2.2 TDesignerAction.Enabled Property**

Indicates whether client controls and menu items are enabled.

```
property Enabled;
```

**Description**

This is inherited property.

See TCustomAction.Enabled for details.

**1.21.1.1.2.3 TDesignerAction.HelpContext Property**

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

```
property HelpContext;
```

**Description**

This is inherited property.

See TCustomAction.HelpContext for details.

**1.21.1.1.2.4 TDesignerAction.HelpKeyword Property**

Indicates the help keyword for client controls and menu items.

```
property HelpKeyword;
```

**Description**

This is inherited property.

See TCustomAction.HelpKeyword for details.

**1.21.1.1.2.5 TDesignerAction.HelpType Property**

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

```
property HelpType;
```

**Description**

This is inherited property.

See TCustomAction.HelpType for details.

**1.21.1.1.2.6 TDesignerAction.Hint Property**

Indicates the Help hint for client controls and menu items.

```
property Hint;
```

**Description**

This is inherited property.

See TCustomAction.Hint for details.

**1.21.1.1.2.7 TDesignerAction.ImageIndex Property**

Indicates the ImageIndex property for client controls and menu items.

```
property ImageIndex;
```

**Description**

This is inherited property.

See TCustomAction.ImageIndex for details.

**1.21.1.1.2.8 TDesignerAction.OnExecute Property**

Occurs when the client event that is linked to it fires.

```
property OnExecute;
```

**Description**

This is inherited property.

See TBasicAction.OnExecute for details.

**1.21.1.1.2.9 TDesignerAction.OnHint Property**

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

**property** OnHint;

#### Description

This is inherited property.

See TCustomAction.OnHint for details.

### 1.21.1.1.2.10 TDesignerAction.OnUpdate Property

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

**property** OnUpdate;

#### Description

This is inherited property.

See TBasicAction.OnUpdate for details.

### 1.21.1.1.2.11 TDesignerAction.SecondaryShortCuts Property

Specifies the short cuts (in addition to ShortCut (see page 311)) for triggering clients.

**property** SecondaryShortCuts;

#### Description

This is inherited property.

See TCustomAction.SecondaryShortCuts for details.

### 1.21.1.1.2.12 TDesignerAction.ShortCut Property

Specifies the ShortCut property for client menu items.

**property** ShortCut;

#### Description

This is inherited property.

See TCustomAction.ShortCut for details.

### 1.21.1.1.2.13 TDesignerAction.Visible Property

Specifies the Visible property for client controls and menu items.

**property** Visible;

#### Description

This is inherited property.

See TCustomAction.Visible for details.

## 1.21.1.2 TdsnAlignmentDlg Class

Opens the Alignment dialog box for the current designer

#### Class Hierarchy



TdsnAlignmentDlg = **class**(TDsnSelAction);

#### File

edActns

**Description**

See TAlignmentDlg for details



**Members****TDesignerAction Methods**

TDesignerAction Methods	Description
 Update  see page 309	Indicates whether the action updates itself.




























**TDsnSelAction Class**

TDsnSelAction Class	Description
 Update  see page 330	Checks if there are selected controls for the active designer




**TdsnAlignmentDlg Class**

TdsnAlignmentDlg Class	Description
 Execute  see page 313	Invokes Alignment dialog and aligns selected components correspondingly for the active designer

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption  see page 309	Represents the caption of client controls and menu items.
 Enabled  see page 309	Indicates whether client controls and menu items are enabled.
 HelpContext  see page 309	Indicates the help context ID for client controls and menu items.
 HelpKeyword  see page 310	Indicates the help keyword for client controls and menu items.
 HelpType  see page 310	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint  see page 310	Indicates the Help hint for client controls and menu items.
 ImageIndex  see page 310	Indicates the ImageIndex property for client controls and menu items.
 OnExecute  see page 310	Occurs when the client event that is linked to it fires.
 OnHint  see page 310	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate  see page 311	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts  see page 311	Specifies the short cuts (in addition to ShortCut  see page 311)) for triggering clients.
 ShortCut  see page 311	Specifies the ShortCut property for client menu items.
 Visible  see page 311	Specifies the Visible property for client controls and menu items.

**Legend**

	Method
	virtual
	Property



**TDesignerAction Methods**

TDesignerAction Methods	Description
 Update  see page 309	Indicates whether the action updates itself.









**TDsnSelAction Class**




















TDsnSelAction Class	Description
 Update  see page 330	Checks if there are selected controls for the active designer

**TdsnAlignmentDlg Class**

TdsnAlignmentDlg Class	Description
 Execute  see page 313	Invokes Alignment dialog and aligns selected components correspondingly for the active designer

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption  see page 309	Represents the caption of client controls and menu items.
 Enabled  see page 309	Indicates whether client controls and menu items are enabled.
 HelpContext  see page 309	Indicates the help context ID for client controls and menu items.
 HelpKeyword  see page 310	Indicates the help keyword for client controls and menu items.

 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

## 1.21.1.2.1 TdsnAlignmentDlg Methods

### 1.21.1.2.1.1 TdsnAlignmentDlg.Execute Method

Invokes Alignment dialog and aligns selected components correspondingly for the active designer

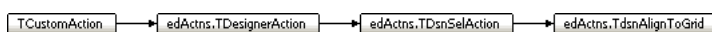
```
function Execute: Boolean; override;
```

#### Description

## 1.21.1.3 TdsnAlignToGrid Class

Performs Align To Grid action for the active designer

#### Class Hierarchy



```
TdsnAlignToGrid = class(TDsnSelAction);
```

#### File

edActns

#### Description

Align to Grid command aligns the selected components to the closest grid point.

#### Members

#### TDesignerAction Methods

TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.









#### TDsnSelAction Class










TDsnSelAction Class	Description
 Update (  see page 330)	Checks if there are selected controls for the active designer

#### TdsnAlignToGrid Class




TdsnAlignToGrid Class	Description
 Execute (  see page 314)	Executes Align To Grid action for the active designer

#### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.

 HelpType ( <a href="#">see page 310</a> )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint ( <a href="#">see page 310</a> )	Indicates the Help hint for client controls and menu items.
 ImageIndex ( <a href="#">see page 310</a> )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute ( <a href="#">see page 310</a> )	Occurs when the client event that is linked to it fires.
 OnHint ( <a href="#">see page 310</a> )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate ( <a href="#">see page 311</a> )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts ( <a href="#">see page 311</a> )	Specifies the short cuts (in addition to ShortCut ( <a href="#">see page 311</a> )) for triggering clients.
 ShortCut ( <a href="#">see page 311</a> )	Specifies the ShortCut property for client menu items.
 Visible ( <a href="#">see page 311</a> )	Specifies the Visible property for client controls and menu items.

## Legend

	Method
	virtual
	Property

## TDesignerAction Methods

TDesignerAction Methods	Description
 Update ( <a href="#">see page 309</a> )	Indicates whether the action updates itself.














## TDsnSelAction Class

TDsnSelAction Class	Description
 Update ( <a href="#">see page 330</a> )	Checks if there are selected controls for the active designer

## TdsnAlignToGrid Class

TdsnAlignToGrid Class	Description
 Execute ( <a href="#">see page 314</a> )	Executes Align To Grid action for the active designer

## TDesignerAction Properties

TDesignerAction Properties	Description
 Caption ( <a href="#">see page 309</a> )	Represents the caption of client controls and menu items.
 Enabled ( <a href="#">see page 309</a> )	Indicates whether client controls and menu items are enabled.
 HelpContext ( <a href="#">see page 309</a> )	Indicates the help context ID for client controls and menu items.
 HelpKeyword ( <a href="#">see page 310</a> )	Indicates the help keyword for client controls and menu items.
 HelpType ( <a href="#">see page 310</a> )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint ( <a href="#">see page 310</a> )	Indicates the Help hint for client controls and menu items.
 ImageIndex ( <a href="#">see page 310</a> )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute ( <a href="#">see page 310</a> )	Occurs when the client event that is linked to it fires.
 OnHint ( <a href="#">see page 310</a> )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate ( <a href="#">see page 311</a> )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts ( <a href="#">see page 311</a> )	Specifies the short cuts (in addition to ShortCut ( <a href="#">see page 311</a> )) for triggering clients.
 ShortCut ( <a href="#">see page 311</a> )	Specifies the ShortCut property for client menu items.
 Visible ( <a href="#">see page 311</a> )	Specifies the Visible property for client controls and menu items.

## 1.21.1.3.1 TdsnAlignToGrid Methods

### 1.21.1.3.1.1 TdsnAlignToGrid.Execute Method

Executes Align To Grid action for the active designer

```
function Execute: Boolean; override;
```

#### Description

Align to Grid command aligns the selected components to the closest grid point.

## 1.21.1.4 TdsnBringToFront Class

Performs Bring To Front action for the active designer

### Class Hierarchy



```
TdsnBringToFront = class(TDsnSelAction);
```

### File

edActns

### Description

Bring to Front command moves a selected component in front of all other components on the form. This is called changing the component's z-order.

### Notes

The Bring to Front and Send to Back commands do not work if you are combining windowed and non-windowed controls. For example, you cannot change the z-order of a label in relation to a button.

### Members

#### TDesignerAction Methods

TDesignerAction Methods	Description
 Update <a href="#">(see page 309)</a>	Indicates whether the action updates itself.














#### TDsnSelAction Class

TDsnSelAction Class	Description
 Update <a href="#">(see page 330)</a>	Checks if there are selected controls for the active designer




#### TdsnBringToFront Class

TdsnBringToFront Class	Description
 Execute <a href="#">(see page 316)</a>	Executes Bring To Front action for the active designer

### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption <a href="#">(see page 309)</a>	Represents the caption of client controls and menu items.
 Enabled <a href="#">(see page 309)</a>	Indicates whether client controls and menu items are enabled.
 HelpContext <a href="#">(see page 309)</a>	Indicates the help context ID for client controls and menu items.
 HelpKeyword <a href="#">(see page 310)</a>	Indicates the help keyword for client controls and menu items.
 HelpType <a href="#">(see page 310)</a>	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint <a href="#">(see page 310)</a>	Indicates the Help hint for client controls and menu items.
 ImageIndex <a href="#">(see page 310)</a>	Indicates the ImageIndex property for client controls and menu items.
 OnExecute <a href="#">(see page 310)</a>	Occurs when the client event that is linked to it fires.
 OnHint <a href="#">(see page 310)</a>	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate <a href="#">(see page 311)</a>	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts <a href="#">(see page 311)</a>	Specifies the short cuts (in addition to ShortCut <a href="#">(see page 311)</a> ) for triggering clients.
 ShortCut <a href="#">(see page 311)</a>	Specifies the ShortCut property for client menu items.
 Visible <a href="#">(see page 311)</a>	Specifies the Visible property for client controls and menu items.

### Legend


	Method
	virtual
	Property



**TDesignerAction Methods**

TDesignerAction Methods	Description
 Update ( <a href="#">see page 309</a> )	Indicates whether the action updates itself.














**TDsnSelAction Class**

TDsnSelAction Class	Description
 Update ( <a href="#">see page 330</a> )	Checks if there are selected controls for the active designer

**TdsnBringToFront Class**

TdsnBringToFront Class	Description
 Execute ( <a href="#">see page 316</a> )	Executes Bring To Front action for the active designer

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption ( <a href="#">see page 309</a> )	Represents the caption of client controls and menu items.
 Enabled ( <a href="#">see page 309</a> )	Indicates whether client controls and menu items are enabled.
 HelpContext ( <a href="#">see page 309</a> )	Indicates the help context ID for client controls and menu items.
 HelpKeyword ( <a href="#">see page 310</a> )	Indicates the help keyword for client controls and menu items.
 HelpType ( <a href="#">see page 310</a> )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint ( <a href="#">see page 310</a> )	Indicates the Help hint for client controls and menu items.
 ImageIndex ( <a href="#">see page 310</a> )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute ( <a href="#">see page 310</a> )	Occurs when the client event that is linked to it fires.
 OnHint ( <a href="#">see page 310</a> )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate ( <a href="#">see page 311</a> )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts ( <a href="#">see page 311</a> )	Specifies the short cuts (in addition to Shortcut ( <a href="#">see page 311</a> )) for triggering clients.
 Shortcut ( <a href="#">see page 311</a> )	Specifies the Shortcut property for client menu items.
 Visible ( <a href="#">see page 311</a> )	Specifies the Visible property for client controls and menu items.

**1.21.1.4.1 TdsnBringToFront Methods****1.21.1.4.1.1 TdsnBringToFront.Execute Method**

Executes Bring To Front action for the active designer

```
function Execute: Boolean; override;
```

**Description**

Bring to Front command moves a selected component in front of all other components on the form. This is called changing the component's z-order.

**Notes**

The Bring to Front and Send to Back commands do not work if you are combining windowed and non-windowed controls. For example, you cannot change the z-order of a label in relation to a button.

**1.21.1.5 TdsnCopy Class**

Performs Copy to clipboard action for the active designer.

**Class Hierarchy**

```
TdsnCopy = class(TEditCopy);
```

**File**

edActns

**Description**

Copy command copies the selected components to the windows clipboard.

**1.21.1.6 TdsnCreationOrderDlg Class**

Opens the Creation Order dialog box for the active designer.

**Class Hierarchy**

```
TdsnCreationOrderDlg = class(TDesignerAction);
```

**File**

edActns



**Description**

This dialog box specifies the order in which your application will create nonvisual components.




























**Members****TDesignerAction Methods**

TDesignerAction Methods	Description
 Update  see page 309	Indicates whether the action updates itself.




**TdsnCreationOrderDlg Class**

TdsnCreationOrderDlg Class	Description
 Execute  see page 318	Executes opening the Creation Order dialog box for the active designer.

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption  see page 309	Represents the caption of client controls and menu items.
 Enabled  see page 309	Indicates whether client controls and menu items are enabled.
 HelpContext  see page 309	Indicates the help context ID for client controls and menu items.
 HelpKeyword  see page 310	Indicates the help keyword for client controls and menu items.
 HelpType  see page 310	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint  see page 310	Indicates the Help hint for client controls and menu items.
 ImageIndex  see page 310	Indicates the ImageIndex property for client controls and menu items.
 OnExecute  see page 310	Occurs when the client event that is linked to it fires.
 OnHint  see page 310	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate  see page 311	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts  see page 311	Specifies the short cuts (in addition to ShortCut  see page 311)) for triggering clients.
 ShortCut  see page 311	Specifies the ShortCut property for client menu items.
 Visible  see page 311	Specifies the Visible property for client controls and menu items.



**Legend**

	Method
	virtual
	Property




























**TDesignerAction Methods**

TDesignerAction Methods	Description
 Update  see page 309	Indicates whether the action updates itself.

**TdsnCreationOrderDlg Class**

TdsnCreationOrderDlg Class	Description
 Execute  see page 318	Executes opening the Creation Order dialog box for the active designer.

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

**1.21.1.6.1 TdsnCreationOrderDlg Methods****1.21.1.6.1.1 TdsnCreationOrderDlg.Execute Method**

Executes opening the Creation Order dialog box for the active designer.

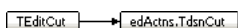
```
function Execute: Boolean; override;
```

**Description**

This dialog box specifies the order in which your application will create nonvisual components.

**1.21.1.7 TdsnCut Class**

Performs Cut To Clipboard action for the active designer.

**Class Hierarchy**

```
TdsnCut = class(TEditCut);
```

**File**

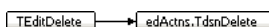
edActns

**Description**

Cut command cuts the selected components to the windows clipboard.

**1.21.1.8 TdsnDelete Class**

Performs Delete action for the current designer

**Class Hierarchy**

```
TdsnDelete = class(TEditDelete);
```

**File**

edActns

**Description**

Delete command deletes the selected components without copying to the clipboard

## 1.21.1.9 TdsnDesignMode Class

Performs switching design mode for the current form with the active designer

**Class Hierarchy**

```
TdsnDesignMode = class(TDesignerAction);
```

**File**

edActns





**Description**

This action switches current form between design and run-time mode.














**Members****TDesignerAction Methods**

TDesignerAction Methods	Description
  Update (see page 309)	Indicates whether the action updates itself.




**TdsnDesignMode Class**

TdsnDesignMode Class	Description
  Execute (see page 320)	Executes switching design mode for the current form with the active designer
  Update (see page 320)	Indicates whether the action updates itself.

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption (see page 309)	Represents the caption of client controls and menu items.
 Enabled (see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (see page 311)	Specifies the short cuts (in addition to ShortCut (see page 311)) for triggering clients.
 ShortCut (see page 311)	Specifies the ShortCut property for client menu items.
 Visible (see page 311)	Specifies the Visible property for client controls and menu items.



**Legend**

	Method
	virtual
	Property














**TDesignerAction Methods**

TDesignerAction Methods	Description
  Update (see page 309)	Indicates whether the action updates itself.

## TdsnDesignMode Class

TdsnDesignMode Class	Description
 Execute ( <a href="#">see page 320</a> )	Executes switching design mode for the current form with the active designer
 Update ( <a href="#">see page 320</a> )	Indicates whether the action updates itself.

## TDesignerAction Properties

TDesignerAction Properties	Description
 Caption ( <a href="#">see page 309</a> )	Represents the caption of client controls and menu items.
 Enabled ( <a href="#">see page 309</a> )	Indicates whether client controls and menu items are enabled.
 HelpContext ( <a href="#">see page 309</a> )	Indicates the help context ID for client controls and menu items.
 HelpKeyword ( <a href="#">see page 310</a> )	Indicates the help keyword for client controls and menu items.
 HelpType ( <a href="#">see page 310</a> )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint ( <a href="#">see page 310</a> )	Indicates the Help hint for client controls and menu items.
 ImageIndex ( <a href="#">see page 310</a> )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute ( <a href="#">see page 310</a> )	Occurs when the client event that is linked to it fires.
 OnHint ( <a href="#">see page 310</a> )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate ( <a href="#">see page 311</a> )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts ( <a href="#">see page 311</a> )	Specifies the short cuts (in addition to ShortCut ( <a href="#">see page 311</a> )) for triggering clients.
 ShortCut ( <a href="#">see page 311</a> )	Specifies the ShortCut property for client menu items.
 Visible ( <a href="#">see page 311</a> )	Specifies the Visible property for client controls and menu items.

## 1.21.1.9.1 TdsnDesignMode Methods

### 1.21.1.9.1.1 TdsnDesignMode.Execute Method

Executes switching design mode for the current form with the active designer

```
function Execute: Boolean; override;
```

#### Description

This action switches current form between design and run-time mode.

### 1.21.1.9.1.2 TdsnDesignMode.Update Method

Indicates whether the action updates itself.

```
function Update: Boolean; override;
```

#### Description

The Update method sets Enable property and Result to True if there is active designer in DsnManager.

## 1.21.1.10 TdsnFlipChildren Class

Performs Flip Children action for the selected controls.

#### Class Hierarchy



```
TdsnFlipChildren = class(TDesignerAction);
```

#### File

edActns


#### Description

Flip Children allows you to reverse the layout of components in the current form to a right-to-left mirror image.

This lets developers quickly change a form created for an audience that reads left to right so that it appears natural in environments where users read from right to left.

## Members




























### TDesignerAction Methods

TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.




### TdsnFlipChildren Class

TdsnFlipChildren Class	Description
 Execute (  see page 322)	Executes Flip Children action for all controls.

### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

## Legend

	Method
	virtual
	Property




























### TDesignerAction Methods

TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.

### TdsnFlipChildren Class

TdsnFlipChildren Class	Description
 Execute (  see page 322)	Executes Flip Children action for all controls.

### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

## 1.21.1.10.1 TdsnFlipChildren Methods

### 1.21.1.10.1.1 TdsnFlipChildren.Execute Method

Executes Flip Children action for all controls.

```
function Execute: Boolean; override;
```

#### Description

## 1.21.1.11 TdsnFlipChildrenAll Class

Performs Flip Children action for all controls.

#### Class Hierarchy



```
TdsnFlipChildrenAll = class(TDesignerAction);
```

#### File

edActns

#### Description

See TdsnFlipChildren (see page 320) for details

#### Members














##### TDesignerAction Methods

TDesignerAction Methods	Description
  Update (see page 309)	Indicates whether the action updates itself.




##### TdsnFlipChildrenAll Class

TdsnFlipChildrenAll Class	Description
  Execute (see page 323)	Executes Flip Children action for all controls.

##### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (see page 309)	Represents the caption of client controls and menu items.
 Enabled (see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (see page 311)	Specifies the short cuts (in addition to ShortCut (see page 311)) for triggering clients.
 ShortCut (see page 311)	Specifies the ShortCut property for client menu items.
 Visible (see page 311)	Specifies the Visible property for client controls and menu items.


#### Legend

	Method
	virtual
	Property














**TDesignerAction Methods**

TDesignerAction Methods	Description
 Update ( <a href="#">see page 309</a> )	Indicates whether the action updates itself.

**TdsnFlipChildrenAll Class**

TdsnFlipChildrenAll Class	Description
 Execute ( <a href="#">see page 323</a> )	Executes Flip Children action for all controls.

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption ( <a href="#">see page 309</a> )	Represents the caption of client controls and menu items.
 Enabled ( <a href="#">see page 309</a> )	Indicates whether client controls and menu items are enabled.
 HelpContext ( <a href="#">see page 309</a> )	Indicates the help context ID for client controls and menu items.
 HelpKeyword ( <a href="#">see page 310</a> )	Indicates the help keyword for client controls and menu items.
 HelpType ( <a href="#">see page 310</a> )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint ( <a href="#">see page 310</a> )	Indicates the Help hint for client controls and menu items.
 ImageIndex ( <a href="#">see page 310</a> )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute ( <a href="#">see page 310</a> )	Occurs when the client event that is linked to it fires.
 OnHint ( <a href="#">see page 310</a> )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate ( <a href="#">see page 311</a> )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts ( <a href="#">see page 311</a> )	Specifies the short cuts (in addition to ShortCut ( <a href="#">see page 311</a> )) for triggering clients.
 ShortCut ( <a href="#">see page 311</a> )	Specifies the ShortCut property for client menu items.
 Visible ( <a href="#">see page 311</a> )	Specifies the Visible property for client controls and menu items.

**1.21.1.11.1 TdsnFlipChildrenAll Methods****1.21.1.11.1.1 TdsnFlipChildrenAll.Execute Method**

Executes Flip Children action for all controls.

```
function Execute: Boolean; override;
```

**Description****1.21.1.12 TdsnGroupControls Class**

Group Controls action for the active designer.

**Class Hierarchy**

```
TdsnGroupControls = class(TDsnSelAction);
```

**File**

edActns

**Description**

The Group command groups the selected controls. Grouped controls are selected together, which makes group operations easier.

**Members****TDesignerAction Methods**







TDesignerAction Methods	Description
 Update ( <a href="#">see page 309</a> )	Indicates whether the action updates itself.






























**TDsnSelAction Class**

TDsnSelAction Class	Description
  Update (  see page 330)	Checks if there are selected controls for the active designer




**TdsnGroupControls Class**

TdsnGroupControls Class	Description
  Execute (  see page 325)	Executes Group Controls action for the active designer
  Update (  see page 325)	Checks if there are selected controls for the active designer

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

**Legend**

	Method
	virtual
	Property





**TDesignerAction Methods**

TDesignerAction Methods	Description
  Update (  see page 309)	Indicates whether the action updates itself.
























**TDsnSelAction Class**





TDsnSelAction Class	Description
  Update (  see page 330)	Checks if there are selected controls for the active designer

**TdsnGroupControls Class**

TdsnGroupControls Class	Description
  Execute (  see page 325)	Executes Group Controls action for the active designer
  Update (  see page 325)	Checks if there are selected controls for the active designer

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.

 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

## 1.21.1.12.1 TdsnGroupControls Methods

### 1.21.1.12.1.1 TdsnGroupControls.Execute Method

Executes Group Controls action for the active designer

```
function Execute: Boolean; override;
```

#### Description

The Group command groups the selected controls. Grouped controls are selected together, which makes group operations easier.

### 1.21.1.12.1.2 TdsnGroupControls.Update Method

Checks if there are selected controls for the active designer

```
function Update: Boolean; override;
```

#### Description

## 1.21.1.13 TdsnLockControls Class

Performs "Lock Controls" action for the active designer

#### Class Hierarchy



```
TdsnLockControls = class(TDesignerAction);
```

#### File

edActns

#### Description



The Lock command locks/unlocks the selected components to prevent them from changing properties

#### Members













#### TDesignerAction Methods
















TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.

#### TdsnLockControls Class




TdsnLockControls Class	Description
 Execute (  see page 326)	Executes Lock Controls action for the active designer
 Update (  see page 326)	Indicates whether the action updates itself.

#### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.

 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.





## Legend

	Method
	virtual
	Property




























## TDesignerAction Methods

TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.

## TdsnLockControls Class

TdsnLockControls Class	Description
 Execute (  see page 326)	Executes Lock Controls action for the active designer
 Update (  see page 326)	Indicates whether the action updates itself.

## TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

## 1.21.1.13.1 TdsnLockControls Methods

### 1.21.1.13.1.1 TdsnLockControls.Execute Method

Executes Lock Controls action for the active designer

```
function Execute: Boolean; override;
```

#### Description

The Lock command locks/unlocks the selected components to prevent them from changing properties

### 1.21.1.13.1.2 TdsnLockControls.Update Method

Indicates whether the action updates itself.

```
function Update: Boolean; override;
```

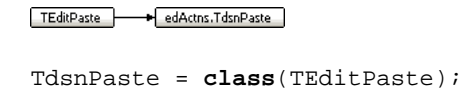
#### Description

The Update method sets Enable property and Result to True if there is active designer in DsnManager.

### 1.21.1.14 TdsnPaste Class

Perform Paste action for the current designer

**Class Hierarchy**



**File**

edActns

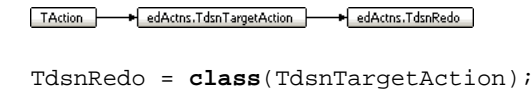
**Description**

Paste command inserts the contents of the clipboard into the current form for the active designer

### 1.21.1.15 TdsnRedo Class

Repeat last undone operation.

**Class Hierarchy**



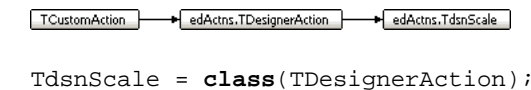
**File**

edActns

### 1.21.1.16 TdsnScale Class

Opens the Scale dialog box for the active designer

**Class Hierarchy**



**File**

edActns

**Description**


Use this dialog box to proportionally resize all the components on the current form.

**Members**



**TDesignerAction Methods**
























TDesignerAction Methods	Description
 Update (see page 309)	Indicates whether the action updates itself.

**TdsnScale Class**




TdsnScale Class	Description
 Execute (see page 328)	Opens the Scale dialog box for the active designer

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption (see page 309)	Represents the caption of client controls and menu items.
 Enabled (see page 309)	Indicates whether client controls and menu items are enabled.

 HelpContext (  )	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  )	Indicates the help keyword for client controls and menu items.
 HelpType (  )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  )	Indicates the Help hint for client controls and menu items.
 ImageIndex (  )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  )	Occurs when the client event that is linked to it fires.
 OnHint (  )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  )	Specifies the short cuts (in addition to ShortCut (  ) for triggering clients.
 ShortCut (  )	Specifies the ShortCut property for client menu items.
 Visible (  )	Specifies the Visible property for client controls and menu items.

## Legend

	Method
	virtual
	Property




























## TDesignerAction Methods

TDesignerAction Methods	Description
 Update (  )	Indicates whether the action updates itself.

## TdsnScale Class

TdsnScale Class	Description
 Execute (  )	Opens the Scale dialog box for the active designer

## TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  )	Represents the caption of client controls and menu items.
 Enabled (  )	Indicates whether client controls and menu items are enabled.
 HelpContext (  )	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  )	Indicates the help keyword for client controls and menu items.
 HelpType (  )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  )	Indicates the Help hint for client controls and menu items.
 ImageIndex (  )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  )	Occurs when the client event that is linked to it fires.
 OnHint (  )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  )	Specifies the short cuts (in addition to ShortCut (  ) for triggering clients.
 ShortCut (  )	Specifies the ShortCut property for client menu items.
 Visible (  )	Specifies the Visible property for client controls and menu items.

## 1.21.1.16.1 TdsnScale Methods

### 1.21.1.16.1.1 TdsnScale.Execute Method

Opens the Scale dialog box for the active designer

```
function Execute: Boolean; override;
```

#### Description

Use this dialog box to proportionally resize all the components on the current form.

## 1.21.1.17 TDsnSelAction Class

Base class of designer actions that require not empty selection

### Class Hierarchy



```
TDsnSelAction = class(TDesignerAction);
```

### File

edActns


### Description

### Members




























#### TDesignerAction Methods

TDesignerAction Methods	Description
 Update  see page 309	Indicates whether the action updates itself.




#### TDsnSelAction Class

TDsnSelAction Class	Description
 Update  see page 330	Checks if there are selected controls for the active designer

#### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption  see page 309	Represents the caption of client controls and menu items.
 Enabled  see page 309	Indicates whether client controls and menu items are enabled.
 HelpContext  see page 310	Indicates the help context ID for client controls and menu items.
 HelpKeyword  see page 310	Indicates the help keyword for client controls and menu items.
 HelpType  see page 310	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint  see page 310	Indicates the Help hint for client controls and menu items.
 ImageIndex  see page 310	Indicates the ImageIndex property for client controls and menu items.
 OnExecute  see page 310	Occurs when the client event that is linked to it fires.
 OnHint  see page 310	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate  see page 311	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts  see page 311	Specifies the short cuts (in addition to ShortCut  see page 311)) for triggering clients.
 ShortCut  see page 311	Specifies the ShortCut property for client menu items.
 Visible  see page 311	Specifies the Visible property for client controls and menu items.

### Legend

	Method
	virtual
	Property

#### TDesignerAction Methods













TDesignerAction Methods	Description
 Update  see page 309	Indicates whether the action updates itself.

#### TDsnSelAction Class

TDsnSelAction Class	Description
 Update  see page 330	Checks if there are selected controls for the active designer

#### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption  see page 309	Represents the caption of client controls and menu items.

 Enabled (🔗 see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (🔗 see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (🔗 see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (🔗 see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (🔗 see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (🔗 see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (🔗 see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (🔗 see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (🔗 see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (🔗 see page 311)	Specifies the short cuts (in addition to ShortCut (🔗 see page 311)) for triggering clients.
 ShortCut (🔗 see page 311)	Specifies the ShortCut property for client menu items.
 Visible (🔗 see page 311)	Specifies the Visible property for client controls and menu items.

### 1.21.1.17.1 TDsnSelAction Methods

#### 1.21.1.17.1.1 TDsnSelAction.Update Method

Checks if there are selected controls for the active designer

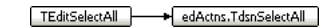
```
function Update: Boolean; override;
```

Description

### 1.21.1.18 TdsnSelectAll Class

Performs Select All action for the active designer

Class Hierarchy



```
TdsnSelectAll = class(TEditSelectAll);
```

File

edActns

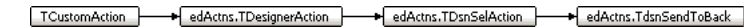
Description

Select All command selects all the items in the active form.

### 1.21.1.19 TdsnSendToBack Class

Performs Send To Back action for the active designer

Class Hierarchy



```
TdsnSendToBack = class(TDsnSelAction);
```

File

edActns

Description

Description

Send To Back command moves a selected component behind all other components on the form.This is called changing the

component's z-order.

## Notes

The Send to Back and Bring to Front commands do not work if you are combining windowed and non-windowed controls. For example, you cannot change the z-order of a label in relation to a button.

## Members

### TDesignerAction Methods

TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.




























### TDsnSelAction Class

TDsnSelAction Class	Description
 Update (  see page 330)	Checks if there are selected controls for the active designer




### TdsnSendToBack Class

TdsnSendToBack Class	Description
 Execute (  see page 332)	Executes Send To Back action for the active designer

### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

## Legend

	Method
	virtual
	Property


### TDesignerAction Methods

TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.









### TDsnSelAction Class

TDsnSelAction Class	Description
 Update (  see page 330)	Checks if there are selected controls for the active designer




















### TdsnSendToBack Class

TdsnSendToBack Class	Description
 Execute (  see page 332)	Executes Send To Back action for the active designer

### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.



 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

## 1.21.1.19.1 TdsnSendToBack Methods

### 1.21.1.19.1.1 TdsnSendToBack.Execute Method

Executes Send To Back action for the active designer

```
function Execute: Boolean; override;
```

#### Description

## 1.21.1.20 TdsnShowTabOrder Class

Set show tab order design mode.

#### Class Hierarchy



```
TdsnShowTabOrder = class(TDesignerAction);
```

#### File

edActns

#### Description





Shows tab order icons over children controls of the selected control. Click on the controls changes their tab order.

#### Members













#### TDesignerAction Methods
















TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.

#### TdsnShowTabOrder Class




TdsnShowTabOrder Class	Description
 Execute (  see page 333)	Shows tab order icons over children controls of the selected control. Click on the controls changes their tab order.
 Update (  see page 333)	Indicates whether the action updates itself.

#### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.

 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.





## Legend

	Method
	virtual
	Property




























## TDesignerAction Methods

TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.

## TdsnShowTabOrder Class

TdsnShowTabOrder Class	Description
 Execute (  see page 333)	Shows tab order icons over children controls of the selected control. Click on the controls changes their tab order.
 Update (  see page 333)	Indicates whether the action updates itself.

## TDesignerAction Properties

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

## 1.21.1.20.1 TdsnShowTabOrder Methods

### 1.21.1.20.1.1 TdsnShowTabOrder.Execute Method

Shows tab order icons over children controls of the selected control. Click on the controls changes their tab order.

```
function Execute: Boolean; override;
```

### 1.21.1.20.1.2 TdsnShowTabOrder.Update Method

Indicates whether the action updates itself.

```
function Update: Boolean; override;
```

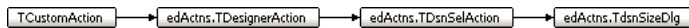
## Description

The Update method sets Enable property and Result to True if there is active designer in DsnManager.

## 1.21.1.21 TdsnSizeDlg Class

Opens the Size dialog box.

### Class Hierarchy



```
TdsnSizeDlg = class(TDsnSelAction);
```

### File

edActns

### Description




Use this dialog box to resize multiple components to be exactly the same height or width.

### Members

#### TDesignerAction Methods

TDesignerAction Methods	Description
  Update  see page 309	Indicates whether the action updates itself.




























#### TDsnSelAction Class

TDsnSelAction Class	Description
  Update  see page 330	Checks if there are selected controls for the active designer




#### TdsnSizeDlg Class

TdsnSizeDlg Class	Description
  Execute  see page 335	Opens the Size dialog box.

#### TDesignerAction Properties

TDesignerAction Properties	Description
 Caption  see page 309	Represents the caption of client controls and menu items.
 Enabled  see page 309	Indicates whether client controls and menu items are enabled.
 HelpContext  see page 309	Indicates the help context ID for client controls and menu items.
 HelpKeyword  see page 310	Indicates the help keyword for client controls and menu items.
 HelpType  see page 310	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint  see page 310	Indicates the Help hint for client controls and menu items.
 ImageIndex  see page 310	Indicates the ImageIndex property for client controls and menu items.
 OnExecute  see page 310	Occurs when the client event that is linked to it fires.
 OnHint  see page 310	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate  see page 311	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts  see page 311	Specifies the short cuts (in addition to ShortCut  see page 311)) for triggering clients.
 ShortCut  see page 311	Specifies the ShortCut property for client menu items.
 Visible  see page 311	Specifies the Visible property for client controls and menu items.

### Legend

	Method
	virtual
	Property

#### TDesignerAction Methods

TDesignerAction Methods	Description
  Update  see page 309	Indicates whether the action updates itself.




























#### TDsnSelAction Class

TDsnSelAction Class	Description
  Update  see page 330	Checks if there are selected controls for the active designer

**TdsnSizeDlg Class**

TdsnSizeDlg Class	Description
 Execute (  see page 335)	Opens the Size dialog box.

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 310)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

**1.21.1.21.1 TdsnSizeDlg Methods****1.21.1.21.1.1 TdsnSizeDlg.Execute Method**

Opens the Size dialog box.

```
function Execute: Boolean; override;
```

**Description**

Use this dialog box to resize multiple components to be exactly the same height or width.

**1.21.1.22 TdsnTabOrderDlg Class**

Opens the Edit Tab Order dialog box

**Class Hierarchy**

```
TdsnTabOrderDlg = class(TDesignerAction);
```

**File**

edActns

**Description**

Use this dialog box to modify the tab order of the components on the form or within the selected component if that component contains other components.




























**Members****TDesignerAction Methods**

TDesignerAction Methods	Description
 Update (  see page 309)	Indicates whether the action updates itself.




**TdsnTabOrderDlg Class**

TdsnTabOrderDlg Class	Description
 Execute (  see page 336)	Opens the Edit Tab Order dialog box

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

**Legend**

	Method
	virtual
	Property




























**TDesignerAction Methods**

TDesignerAction Methods	Description
  Update (  see page 309)	Indicates whether the action updates itself.

**TdsnTabOrderDlg Class**

TdsnTabOrderDlg Class	Description
  Execute (  see page 336)	Opens the Edit Tab Order dialog box

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption (  see page 309)	Represents the caption of client controls and menu items.
 Enabled (  see page 309)	Indicates whether client controls and menu items are enabled.
 HelpContext (  see page 309)	Indicates the help context ID for client controls and menu items.
 HelpKeyword (  see page 310)	Indicates the help keyword for client controls and menu items.
 HelpType (  see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint (  see page 310)	Indicates the Help hint for client controls and menu items.
 ImageIndex (  see page 310)	Indicates the ImageIndex property for client controls and menu items.
 OnExecute (  see page 310)	Occurs when the client event that is linked to it fires.
 OnHint (  see page 310)	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate (  see page 311)	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts (  see page 311)	Specifies the short cuts (in addition to ShortCut (  see page 311)) for triggering clients.
 ShortCut (  see page 311)	Specifies the ShortCut property for client menu items.
 Visible (  see page 311)	Specifies the Visible property for client controls and menu items.

**1.21.1.22.1 TdsnTabOrderDlg Methods****1.21.1.22.1.1 TdsnTabOrderDlg.Execute Method**

Opens the Edit Tab Order dialog box

```
function Execute: Boolean; override;
```

**Description**

Use this dialog box to modify the tab order of the components on the form or within the selected component if that

component contains other components.

### 1.21.1.23 TdsnTargetAction Class

Designer target actions. Determines whether target is managed by designer.

#### Class Hierarchy



```
TdsnTargetAction = class(TAction);
```

#### File

edActns

### 1.21.1.24 TdsnTextEditMode Class

Toggles TextEditMode of the active designer.

#### Class Hierarchy



```
TdsnTextEditMode = class(TDesignerAction);
```

#### File

edActns

#### Description

#### Members

##### TDesignerAction Methods

TDesignerAction Methods	Description
Update (see page 309)	Indicates whether the action updates itself.




##### TdsnTextEditMode Class

TdsnTextEditMode Class	Description
Execute (see page 338)	Toggles TextEditMode of the active designer.
Update (see page 338)	Indicates whether the action updates itself.


##### TDesignerAction Properties

TDesignerAction Properties	Description
Caption (see page 309)	Represents the caption of client controls and menu items.
Enabled (see page 309)	Indicates whether client controls and menu items are enabled.
HelpContext (see page 309)	Indicates the help context ID for client controls and menu items.
HelpKeyword (see page 310)	Indicates the help keyword for client controls and menu items.
HelpType (see page 310)	Indicates the mechanism for client controls and menu items to use when invoking help.
Hint (see page 310)	Indicates the Help hint for client controls and menu items.
ImageIndex (see page 310)	Indicates the ImageIndex property for client controls and menu items.
OnExecute (see page 310)	Occurs when the client event that is linked to it fires.
OnHint (see page 310)	Occurs when the mouse pauses over a client control or menu item.
OnUpdate (see page 311)	Occurs when the application is idle or when the action list updates.
SecondaryShortCuts (see page 311)	Specifies the short cuts (in addition to ShortCut (see page 311)) for triggering clients.
ShortCut (see page 311)	Specifies the ShortCut property for client menu items.
Visible (see page 311)	Specifies the Visible property for client controls and menu items.





**Legend**

	Method
	virtual
	Property




























**TDesignerAction Methods**

TDesignerAction Methods	Description
 Update  see page 309	Indicates whether the action updates itself.

**TdsnTextEditMode Class**

TdsnTextEditMode Class	Description
 Execute  see page 338	Toggles TextEditMode of the active designer.
 Update  see page 338	Indicates whether the action updates itself.

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption  see page 309	Represents the caption of client controls and menu items.
 Enabled  see page 309	Indicates whether client controls and menu items are enabled.
 HelpContext  see page 309	Indicates the help context ID for client controls and menu items.
 HelpKeyword  see page 310	Indicates the help keyword for client controls and menu items.
 HelpType  see page 310	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint  see page 310	Indicates the Help hint for client controls and menu items.
 ImageIndex  see page 310	Indicates the ImageIndex property for client controls and menu items.
 OnExecute  see page 310	Occurs when the client event that is linked to it fires.
 OnHint  see page 310	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate  see page 311	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts  see page 311	Specifies the short cuts (in addition to ShortCut  see page 311)) for triggering clients.
 ShortCut  see page 311	Specifies the ShortCut property for client menu items.
 Visible  see page 311	Specifies the Visible property for client controls and menu items.

**1.21.1.24.1 TdsnTextEditMode Methods****1.21.1.24.1.1 TdsnTextEditMode.Execute Method**

Toggles TextEditMode of the active designer.

```
function Execute: Boolean; override;
```

**Description****1.21.1.24.1.2 TdsnTextEditMode.Update Method**

Indicates whether the action updates itself.

```
function Update: Boolean; override;
```

**Description**

The Update method sets Enable property and Result to True if there is active designer in DsnManager.

**1.21.1.25 TdsnUndo Class**

Backs out last change in the undo buffer.

**Class Hierarchy**

```
TdsnUndo = class(TEditUndo);
```

**File**

edActns

**Description**

## 1.21.1.26 TdsnUngroupControls Class

Ungroups selected controls.

**Class Hierarchy**

```
TdsnUngroupControls = class(TDsnSelAction);
```

**File**

edActns

**Description**

The Ungroup command removes all groups to which the selected controls belong.



**Members****TDesignerAction Methods**

TDesignerAction Methods	Description
 Update  see page 309	Indicates whether the action updates itself.




























**TDsnSelAction Class**

TDsnSelAction Class	Description
 Update  see page 330	Checks if there are selected controls for the active designer




**TdsnUngroupControls Class**

TdsnUngroupControls Class	Description
 Execute  see page 340	Executes Ungroup Controls action for the active designer

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption  see page 309	Represents the caption of client controls and menu items.
 Enabled  see page 309	Indicates whether client controls and menu items are enabled.
 HelpContext  see page 309	Indicates the help context ID for client controls and menu items.
 HelpKeyword  see page 310	Indicates the help keyword for client controls and menu items.
 HelpType  see page 310	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint  see page 310	Indicates the Help hint for client controls and menu items.
 ImageIndex  see page 310	Indicates the ImageIndex property for client controls and menu items.
 OnExecute  see page 310	Occurs when the client event that is linked to it fires.
 OnHint  see page 310	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate  see page 311	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts  see page 311	Specifies the short cuts (in addition to ShortCut  see page 311)) for triggering clients.
 ShortCut  see page 311	Specifies the ShortCut property for client menu items.
 Visible  see page 311	Specifies the Visible property for client controls and menu items.

**Legend**


	Method
	virtual
	Property



**TDesignerAction Methods**

TDesignerAction Methods	Description
 Update ( <a href="#">see page 309</a> )	Indicates whether the action updates itself.














**TDsnSelAction Class**

TDsnSelAction Class	Description
 Update ( <a href="#">see page 330</a> )	Checks if there are selected controls for the active designer

**TdsnUngroupControls Class**

TdsnUngroupControls Class	Description
 Execute ( <a href="#">see page 340</a> )	Executes Ungroup Controls action for the active designer

**TDesignerAction Properties**

TDesignerAction Properties	Description
 Caption ( <a href="#">see page 309</a> )	Represents the caption of client controls and menu items.
 Enabled ( <a href="#">see page 309</a> )	Indicates whether client controls and menu items are enabled.
 HelpContext ( <a href="#">see page 309</a> )	Indicates the help context ID for client controls and menu items.
 HelpKeyword ( <a href="#">see page 310</a> )	Indicates the help keyword for client controls and menu items.
 HelpType ( <a href="#">see page 310</a> )	Indicates the mechanism for client controls and menu items to use when invoking help.
 Hint ( <a href="#">see page 310</a> )	Indicates the Help hint for client controls and menu items.
 ImageIndex ( <a href="#">see page 310</a> )	Indicates the ImageIndex property for client controls and menu items.
 OnExecute ( <a href="#">see page 310</a> )	Occurs when the client event that is linked to it fires.
 OnHint ( <a href="#">see page 310</a> )	Occurs when the mouse pauses over a client control or menu item.
 OnUpdate ( <a href="#">see page 311</a> )	Occurs when the application is idle or when the action list updates.
 SecondaryShortCuts ( <a href="#">see page 311</a> )	Specifies the short cuts (in addition to ShortCut ( <a href="#">see page 311</a> )) for triggering clients.
 ShortCut ( <a href="#">see page 311</a> )	Specifies the ShortCut property for client menu items.
 Visible ( <a href="#">see page 311</a> )	Specifies the Visible property for client controls and menu items.

**1.21.1.26.1 TdsnUngroupControls Methods****1.21.1.26.1.1 TdsnUngroupControls.Execute Method**

Executes Ungroup Controls action for the active designer

```
function Execute: Boolean; override;
```

**Description**

The Ungroup command removes all groups to which the selected controls belong.

**1.22 edcCmbCombo Namespace****1.22.1 Classes**

The following table lists classes in this documentation.

**Classes**

Class	Description
TComponentCombo ( <a href="#">see page 341</a> )	TComponentCombo represents object inspector's combo box. It contains all components in the form (data module, ...), that is designed by the active designer

## 1.22.1.1 TComponentCombo Class

TComponentCombo represents object inspector's combo box. It contains all components in the form (data module, ...), that is designed by the active designer

### Class Hierarchy



```
TComponentCombo = class(TCustomComboBox, IDesignNotification);
```

### File

edcCmbCombo

### Description

TComponentCombo used for maintaining list of components owned by designed window.















### Members

#### TComponentCombo Methods





TComponentCombo Methods	Description
Change (see page 344)	Determines component's name picked in the list and selects appropriate components on the designed form.
Create (see page 344)	Creates and initializes a TComponentCombo instance.
Destroy (see page 345)	Destroys an instance of TComponentCombo.
DoAddObject (see page 345)	Adds object to drop-down list.
FillObjList (see page 345)	Fills combo box items.
Notification (see page 345)	Forwards notification messages to all owned components.
SetSelection (see page 345)	Sets selection.
UpdateObjectList (see page 345)	Update items.

#### TComponentCombo Properties




TComponentCombo Properties	Description
Align (see page 346)	Determines how the control aligns within its container (parent control).
Anchors (see page 346)	Specifies how the control is anchored to its parent.
AutoCloseUp (see page 347)	Specifies whether the drop-down closes up automatically when the user selects an item.
AutoDropDown (see page 347)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
AutoHint (see page 347)	Specifies whether combobox automatically updates its Hint property.
BiDiMode (see page 347)	Specifies the bi-directional mode for the control.
ClassNameColor (see page 347)	Specifies font color for component's class names.
ClassNameDelim (see page 348)	Specifies delimiter between component name and its class name
Color (see page 348)	Specifies the background color of the control.
Constraints (see page 348)	Specifies the size constraints for the control.
Ctl3D (see page 348)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
Designer (see page 348)	Allows direct linking to the particular designer. Component combo can work also with inactive designer.
DragCursor (see page 348)	Indicates the image used to represent the mouse pointer when the control is being dragged.
DragKind (see page 349)	Specifies whether the control is being dragged normally or for docking.
DragMode (see page 349)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
DropDownCount (see page 349)	Specifies the maximum number of items displayed in the drop-down list.
DropDownWidth (see page 349)	Specifies the width, in pixels, of the drop-down list.
Enabled (see page 349)	Controls whether the control responds to mouse, keyboard, and timer events.
Font (see page 350)	Controls the attributes of text written on or in the control.
ImeMode (see page 350)	Determines the behavior of the input method editor (IME).



 ImeName (see page 350)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
 IncludeContainer (see page 350)	Specifies if root component (for example, Form) will be added to the list.
 ItemHeight (see page 350)	Specifies the height, in pixels, of the items in the drop-down list.
 MaxLength (see page 351)	Specifies the maximum number of characters the user can type into the edit portion.
 NameColor (see page 351)	Specifies font color for component names portion in the list.
 OnClick (see page 351)	Occurs when the user clicks the control.
 OnCloseUp (see page 351)	Occurs when the drop-down list closes up due to some user action.
 OnContextPopup (see page 351)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 352)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 352)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 352)	Occurs when the user drags an object over a control.
 OnDrawItem (see page 353)	Occurs when an item in an owner-draw combo box needs to be displayed.
 OnDropDown (see page 353)	Occurs when the user opens the drop-down list by clicking the arrow at the right of the control.
 OnEndDock (see page 353)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
 OnEndDrag (see page 353)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 353)	Occurs when a control receives the input focus.
 OnExit (see page 354)	Occurs when the input focus shifts away from one control to another.
 OnKeyDown (see page 354)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 354)	Occurs when key pressed.
 OnKeyUp (see page 354)	Occurs when the user releases a key that has been pressed.
 OnMeasureItem (see page 355)	Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.
 OnSelect (see page 355)	Occurs when the user selects an item in the list.
 OnStartDock (see page 355)	Occurs when the user begins to drag a control with a DragKind of dkDock.
 OnStartDrag (see page 355)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 356)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 356)	Determines where a control looks for its color information.
 ParentCtl3D (see page 356)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 356)	Determines where a control looks for its font information.
 ParentShowHint (see page 356)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 356)	Identifies the pop-up menu associated with the control.
 ShowClassName (see page 356)	Specifies whether class names will be displayed in the items
 ShowComponents (see page 357)	Determines if only TControl descendants will be shown in the list
 ShowHint (see page 357)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 Sorted (see page 357)	Determines whether the list portion of the combo box is alphabetized.
 TabOrder (see page 357)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 357)	Determines if the user can tab to a control.
 Text (see page 357)	Specifies the text string that is displayed in the edit box.
 Visible (see page 358)	Determines whether the component appears on screen.

## TComponentCombo Events





TComponentCombo Events	Description
 OnCanAddObject (see page 358)	Occurs to defined whether object Obj can be added to drop-down list.
 OnGetClassName (see page 358)	Allows replacing displayed class name.
 OnGetComponents (see page 358)	Called during updating item list.
 OnSelChanged (see page 358)	Called when selected item (object) has bin changed by selecting from drop-down list.

## Legend

	Method
	protected
	virtual

	Property
	Event





























## TComponentCombo Events

TComponentCombo Events	Description
 OnCanAddObject (see page 358)	Occurs to defined whether object Obj can be added to drop-down list.
 OnGetClassName (see page 358)	Allows replacing displayed class name.
 OnGetComponents (see page 358)	Called during updating item list.
 OnSelChanged (see page 358)	Called when selected item (object) has bin changed by selecting from drop-down list.

## TComponentCombo Methods

TComponentCombo Methods	Description
 Change (see page 344)	Determines component's name picked in the list and selects appropriate components on the designed form.
 Create (see page 344)	Creates and initializes a TComponentCombo instance.
 Destroy (see page 345)	Destroys an instance of TComponentCombo.
 DoAddObject (see page 345)	Adds object to drop-down list.
 FillObjList (see page 345)	Fills combo box items.
 Notification (see page 345)	Forwards notification messages to all owned components.
 SetSelection (see page 345)	Sets selection.
 UpdateObjectList (see page 345)	Update items.

## TComponentCombo Properties

TComponentCombo Properties	Description
 Align (see page 346)	Determines how the control aligns within its container (parent control).
 Anchors (see page 346)	Specifies how the control is anchored to its parent.
 AutoCloseUp (see page 347)	Specifies whether the drop-down closes up automatically when the user selects an item.
 AutoDropDown (see page 347)	Specifies whether the drop-down list drops down automatically in response to user keystrokes.
 AutoHint (see page 347)	Specifies whether combobox automatically updates it's Hint property.
 BiDiMode (see page 347)	Specifies the bi-directional mode for the control.
 ClassNameColor (see page 347)	Specifies font color for component's class names.
 ClassNameDelim (see page 348)	Specifies delimiter between component name and its class name
 Color (see page 348)	Specifies the background color of the control.
 Constraints (see page 348)	Specifies the size constraints for the control.
 Ctl3D (see page 348)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 Designer (see page 348)	Allows direct linking to the particular designer. Component combo can work also with inactive designer.
 DragCursor (see page 348)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 349)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 349)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 DropDownCount (see page 349)	Specifies the maximum number of items displayed in the drop-down list.
 DropDownWidth (see page 349)	Specifies the width, in pixels, of the drop-down list.
 Enabled (see page 349)	Controls whether the control responds to mouse, keyboard, and timer events.
 Font (see page 350)	Controls the attributes of text written on or in the control.
 ImeMode (see page 350)	Determines the behavior of the input method editor (IME).
 ImeName (see page 350)	Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.
 IncludeContainer (see page 350)	Specifies if root component (for example, Form) will be added to the list.
 ItemHeight (see page 350)	Specifies the height, in pixels, of the items in the drop-down list.
 MaxLength (see page 351)	Specifies the maximum number of characters the user can type into the edit portion.
 NameColor (see page 351)	Specifies font color for component names portion in the list.
 OnClick (see page 351)	Occurs when the user clicks the control.
 OnCloseUp (see page 351)	Occurs when the drop-down list closes up due to some user action.
 OnContextPopup (see page 351)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

 OnDbClick (see page 352)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 352)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 352)	Occurs when the user drags an object over a control.
 OnDrawItem (see page 353)	Occurs when an item in an owner-draw combo box needs to be displayed.
 OnDropDown (see page 353)	Occurs when the user opens the drop-down list by clicking the arrow at the right of the control.
 OnEndDock (see page 353)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
 OnEndDrag (see page 353)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 353)	Occurs when a control receives the input focus.
 OnExit (see page 354)	Occurs when the input focus shifts away from one control to another.
 OnKeyDown (see page 354)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 354)	Occurs when key pressed.
 OnKeyUp (see page 354)	Occurs when the user releases a key that has been pressed.
 OnMeasureItem (see page 355)	Occurs when an item in a csOwnerDraw/Variable combo box needs to be redisplayed.
 OnSelect (see page 355)	Occurs when the user selects an item in the list.
 OnStartDock (see page 355)	Occurs when the user begins to drag a control with a DragKind of dkDock.
 OnStartDrag (see page 355)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 356)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 356)	Determines where a control looks for its color information.
 ParentCtl3D (see page 356)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 356)	Determines where a control looks for its font information.
 ParentShowHint (see page 356)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 356)	Identifies the pop-up menu associated with the control.
 ShowClassName (see page 356)	Specifies whether class names will be displayed in the items
 ShowComponents (see page 357)	Determines if only TControl descendants will be shown in the list
 ShowHint (see page 357)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 Sorted (see page 357)	Determines whether the list portion of the combo box is alphabetized.
 TabOrder (see page 357)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 357)	Determines if the user can tab to a control.
 Text (see page 357)	Specifies the text string that is displayed in the edit box.
 Visible (see page 358)	Determines whether the component appears on screen.

### 1.22.1.1.1 TComponentCombo Methods

#### 1.22.1.1.1.1 TComponentCombo.Change Method

Determines component's name picked in the list and selects appropriate components on the designed form.

```
procedure Change; override;
```

#### 1.22.1.1.1.2 TComponentCombo.Create Constructor

Creates and initializes a TComponentCombo instance.

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

#### Description

Use Create to programmatically instantiate a TComponentCombo component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

### 1.22.1.1.1.3 TComponentCombo.Destroy Destructor

Destroys an instance of TComponentCombo.

```
destructor Destroy; override;
```

#### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

### 1.22.1.1.1.4 TComponentCombo.DoAddObject Method

Adds object to drop-down list.

```
procedure DoAddObject(Obj: TObject); dynamic;
```

### 1.22.1.1.1.5 TComponentCombo.FillObjList Method

Fills combo box items.

```
procedure FillObjList;
```

#### Description

Adds all object that can be selected to list. Object reference is saved in Items.Objects[] members.

### 1.22.1.1.1.6 TComponentCombo.Notification Method

Forwards notification messages to all owned components.

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

#### Description

Do not call the Notification method in an application. Notification is called automatically when the component specified by AComponent is about to be inserted or removed, as specified by Operation. By default, components pass along the notification to their owned components, if any.

A component can, if needed, act on the notification that a component is being inserted or removed. For example, if a component has object fields or properties that contain references to other components, it can check the notifications of component removals and invalidate those references as needed.

#### Notes

Notification is not called for components that are freed implicitly (because their Owner is freed).

### 1.22.1.1.1.7 TComponentCombo.SetSelection Method

Sets selection.

```
procedure SetSelection(SelCount: integer; SelObj: TPersistent);
```

#### Description

Uses SelObj to select appropriate item in list if SelCount is 1, otherwise displays number of selected items.

### 1.22.1.1.1.8 TComponentCombo.UpdateObjectList Method

Update items.

```
procedure UpdateObjectList; virtual;
```

#### Description

May be changed by derived classes. By default it calls FillObjList (see page 345).

## 1.22.1.1.2 TComponentCombo Properties

### 1.22.1.1.2.1 TComponentCombo.Align Property

Determines how the control aligns within its container (parent control).

**property** Align;

#### Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to `alLeft`. The value of `alLeft` for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is `alNone`, which means a control remains where it is positioned on a form or panel.

**Tip:** If Align is set to `alClient`, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

**Note:** To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

### 1.22.1.1.2.2 TComponentCombo.Anchors Property

Specifies how the control is anchored to its parent.

**property** Anchors;

#### Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to `[akLeft, akRight]`, the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

**Note:** If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

### 1.22.1.1.2.3 TComponentCombo.AutoCloseUp Property

Specifies whether the drop-down closes up automatically when the user selects an item.

```
property AutoCloseUp;
```

#### Description

When AutoCloseUp is true, the drop-down closes up automatically when the user selects an item.

### 1.22.1.1.2.4 TComponentCombo.AutoDropDown Property

Specifies whether the drop-down list drops down automatically in response to user keystrokes.

```
property AutoDropDown;
```

#### Description

When AutoDropDown is true, the combo box automatically drops down its list when the user starts typing a string while the combo box has focus.

When AutoDropDown is false, the user must explicitly use the drop-down button to drop down the combo box list.

### 1.22.1.1.2.5 TComponentCombo.AutoHint Property

Specifies whether combobox automatically updates its Hint property.

```
property AutoHint: Boolean;
```

#### Description

When AutoHint is True TComponentCombo (see page 341) changes Hint property each time selected object changed.

Hint format:

[Object Path] : [Object Class Name]

### 1.22.1.1.2.6 TComponentCombo.BiDiMode Property

Specifies the bi-directional mode for the control.

```
property BiDiMode;
```

#### Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

### 1.22.1.1.2.7 TComponentCombo.ClassNameColor Property

Specifies font color for component's class names.

```
property ClassNameColor: TColor;
```

#### Description

Set this property to change (see page 344) font color for component's class names in the list.

Default value is clGrayText.



### 1.22.1.1.2.8 TComponentCombo.ClassNameDelim Property

Specifies delimiter between component name and its class name

```
property ClassNameDelim: string;
```

#### Description

Set this property to specify delimiter string between component name and its class name.

Default value is ' '.

### 1.22.1.1.2.9 TComponentCombo.Color Property

Specifies the background color of the control.

```
property Color;
```

#### Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

### 1.22.1.1.2.10 TComponentCombo.Constraints Property

Specifies the size constraints for the control.

```
property Constraints;
```

#### Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

**Warning:** Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

### 1.22.1.1.2.11 TComponentCombo.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

```
property Ctl3D;
```

#### Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

### 1.22.1.1.2.12 TComponentCombo.Designer Property

Allows direct linking to the particular designer. Component combo can work also with inactive designer.

```
property Designer: TzFormDesigner;
```

### 1.22.1.1.2.13 TComponentCombo.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

```
property DragCursor;
```

#### Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

**Note:** To make a custom cursor available for the DragCursor property, see the Cursor property.

#### 1.22.1.1.2.14 TComponentCombo.DragKind Property

Specifies whether the control is being dragged normally or for docking.

**property** DragKind;

##### Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

#### 1.22.1.1.2.15 TComponentCombo.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

**property** DragMode;

##### Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

#### 1.22.1.1.2.16 TComponentCombo.DropDownCount Property

Specifies the maximum number of items displayed in the drop-down list.

**property** DropDownCount;

##### Description

By default, the drop-down list is long enough to contain eight items without requiring the user to scroll to see them all. To make the drop-down list smaller or larger, specify a number larger or smaller than eight as the DropDownCount value.

If the DropDownCount value is larger than the number of items in the Items property, the drop-down list will be just large enough to hold all the possible choices and no more. If the DropDownCount value is smaller than the number of items in the Items property, the drop-down list will display a scroll bar.

#### 1.22.1.1.2.17 TComponentCombo.DropDownWidth Property

Specifies the width, in pixels, of the drop-down list.

**property** DropDownWidth: integer;

##### Description

Use DropDownWidth to customize the width of the drop-down list. If DropDownWidth is 0 (the default), the drop-down list is the same width as the combo box.

#### 1.22.1.1.2.18 TComponentCombo.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

**property** Enabled;

##### Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

### 1.22.1.1.2.19 TComponentCombo.Font Property

Controls the attributes of text written on or in the control.

```
property Font;
```

#### Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

### 1.22.1.1.2.20 TComponentCombo.ImeMode Property

Determines the behavior of the input method editor (IME).

```
property ImeMode;
```

#### Description

Set ImeMode to configure the way an IME processes user keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeMode allows a control to influence the type of conversion performed by the IME so that it is appropriate for the input expected by the control. For example, a control that only accepts numeric input might specify an ImeMode of imClose, as no conversion is necessary for numeric input.

### 1.22.1.1.2.21 TComponentCombo.ImeName Property

Specifies the input method editor (IME) to use for converting keyboard input to Asian language characters.

```
property ImeName;
```

#### Description

Set ImeName to specify which IME to use for converting keystrokes. An IME is a front-end input processor for Asian language characters. The IME hooks all keyboard input, converts it to Asian characters in a conversion window, and sends the converted characters or strings on to the application.

ImeName must specify one of the IMEs that has been installed through the Windows control panel. The property inspector provides a drop-down list of all currently installed IMEs on the system. At runtime, applications can obtain a list of currently installed IMEs from the global Screen variable.

If ImeName specifies an unavailable IME, the IME that was active when the application started is used instead. No exception is generated.

### 1.22.1.1.2.22 TComponentCombo.IncludeContainer Property

Specifies if root component (for example, Form) will be added to the list.

```
property IncludeContainer: Boolean;
```

#### Description

If IncludeContainer is True, root component will be added to the list.

### 1.22.1.1.2.23 TComponentCombo.ItemHeight Property

Specifies the height, in pixels, of the items in the drop-down list.

```
property ItemHeight;
```

**Description**

Use `ItemHeight` to specify the height needed to draw the items in the list when `Style` is set to `csOwnerDrawFixed`. If `Style` is `csOwnerDrawVariable`, `ItemHeight` is the default height for drawing list items. When `Style` is `csOwnerDrawVariable`, `ItemHeight` can be overridden by an `OnMeasureItem` event handler. If `Style` is set to any other value, the `ItemHeight` property is inoperative.

**1.22.1.1.2.24 TComponentCombo.MaxLength Property**

Specifies the maximum number of characters the user can type into the edit portion.

```
property MaxLength;
```

**Description**

Use `MaxLength` to limit the length of the items in the combo box. `MaxLength` limits the length of the string in the edit portion of the combo box. If `MaxLength` is less than the length of the current value of the `Text` property, this string is truncated.

If `MaxLength` is `-1`, there is no limit to the length of the strings that can appear in the combo box.

**1.22.1.1.2.25 TComponentCombo.NameColor Property**

Specifies font color for component names portion in the list.

```
property NameColor: TColor;
```

**Description**

Set this property to change (see page 344) font color for component names in the list.

Default value is `clWindowText`.

**1.22.1.1.2.26 TComponentCombo.OnClick Property**

Occurs when the user clicks the control.

```
property OnClick;
```

**1.22.1.1.2.27 TComponentCombo.OnCloseUp Property**

Occurs when the drop-down list closes up due to some user action.

```
property OnCloseUp;
```

**Description**

Write an `OnCloseUp` event handler to implement special processing that needs to occur when the drop-down list closes up. For example, an `OnCloseUp` event handler can check whether the user changed the selected item while the list was dropped down and respond accordingly.

**1.22.1.1.2.28 TComponentCombo.OnContextPopup Property**

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

```
property OnContextPopup;
```

**Description**

The `OnContextPopup` handler is called when the user uses the mouse or keyboard to request a popup menu. The `OnContextPopup` event is generated by a `WM_CONTEXTMENU` message, which is itself generated by the user clicking the right mouse button or by pressing `SHIFT+F10` or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the `PopupMenu` property is not

set) or if the `AutoPopup` property of the control's associated popup menu is false. However, the `OnContextPopup` can also be used to override the automatic context menu that appears when the control has an associated popup menu with an `AutoPopup` property of true. In this last case, if the event handler displays its own menu, it should set the `Handled` parameter to true to suppress the default context menu.

The handler's `MousePos` parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, `MousePos` is (-1,-1).

**Note:** Parent controls receive an `OnContextPopup` event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an `OnContextPopup` event after the child control. As a result, when parent controls do not set `Handled` to true in an `OnContextPopup` event handler, the event handler may be called multiple times for each context menu invocation.

#### 1.22.1.1.2.29 TComponentCombo.OnDbClick Property

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

```
property OnDbClick;
```

##### Description

Use the `OnDbClick` event to respond to mouse double-clicks.

#### 1.22.1.1.2.30 TComponentCombo.OnDragDrop Property

Occurs when the user drops an object being dragged.

```
property OnDragDrop;
```

##### Description

Use the `OnDragDrop` event handler to specify what happens when the user drops an object. The `Source` parameter of the `OnDragDrop` event is the object being dropped, and the `Sender` is the control the object is being dropped on. The `X` and `Y` parameters are the coordinates of the mouse positioned over the control.

#### 1.22.1.1.2.31 TComponentCombo.OnDragOver Property

Occurs when the user drags an object over a control.

```
property OnDragOver;
```

##### Description

Use an `OnDragOver` event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the `OnDragOver` event handler, change the `Accept` parameter to false to reject the dragged object. Leave `Accept` as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the `DragCursor` property for the control before the `OnDragOver` event occurs.

The `Source` is the object being dragged, the `Sender` is the potential drop or dock site, and `X` and `Y` are screen coordinates in pixels. The `State` parameter specifies how the dragged object is moving over the control.

**Note:** Within the `OnDragOver` event handler, the `Accept` parameter defaults to true. However, if an `OnDragOver` event

handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

#### 1.22.1.1.2.32 TComponentCombo.OnDrawItem Property

Occurs when an item in an owner-draw combo box needs to be displayed.

**property** OnDrawItem;

##### Description

Write an OnDrawItem event handler to draw the items in the drop-down list of an owner-draw combo box. An OnDrawItem event handler can add graphic elements to the list items, or replace the list item text by graphics.

Draw the items on the Canvas using the coordinates provided by the Rect parameter. OnDrawItem occurs only if Style is set to csOwnerDrawFixed or csOwnerDrawVariable.

If an OnDrawItem event handler is not provided, the combo box fills the Rect parameter with the current brush and writes the text value of the item specified by the Index parameter.

#### 1.22.1.1.2.33 TComponentCombo.OnDropDown Property

Occurs when the user opens the drop-down list by clicking the arrow at the right of the control.

**property** OnDropDown;

##### Description

Write an OnDropDown event handler to implement special processing that needs to occur only when the drop-down list is activated.

##### Notes

OnDropDown never occurs if the combo box does not include any items.

#### 1.22.1.1.2.34 TComponentCombo.OnEndDock Property

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

**property** OnEndDock;

##### Description

Use OnEndDock to specify actions or special processing that when a drag-and-dock operation stops.

#### 1.22.1.1.2.35 TComponentCombo.OnEndDrag Property

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

**property** OnEndDrag;

##### Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

#### 1.22.1.1.2.36 TComponentCombo.OnEnter Property

Occurs when a control receives the input focus.

**property** OnEnter;

##### Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that

includes the control.

#### 1.22.1.1.2.37 TComponentCombo.OnExit Property

Occurs when the input focus shifts away from one control to another.

**property** OnExit;

##### Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

#### 1.22.1.1.2.38 TComponentCombo.OnKeyDown Property

Occurs when a user presses any key while the control has focus.

**property** OnKeyDown;

##### Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys, and pressed mouse buttons.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

#### 1.22.1.1.2.39 TComponentCombo.OnKeyPress Property

Occurs when key pressed.

**property** OnKeyPress;

##### Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

#### 1.22.1.1.2.40 TComponentCombo.OnKeyUp Property

Occurs when the user releases a key that has been pressed.

**property** OnKeyUp;

##### Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

#### 1.22.1.1.2.41 TComponentCombo.OnMeasureItem Property

Occurs when an item in a csOwnerDrawVariable combo box needs to be redisplayed.

```
property OnMeasureItem;
```

##### Description

When Style is set to csOwnerDrawVariable, the OnMeasureItem event precedes OnDrawItem. Write an OnMeasureItem event handler to specify the height, in pixels, needed to draw an item in the drop-down list.

#### 1.22.1.1.2.42 TComponentCombo.OnSelect Property

Occurs when the user selects an item in the list.

```
property OnSelect;
```

##### Description

Write an OnSelect event handler to respond when the user selects a new item in the list.

#### 1.22.1.1.2.43 TComponentCombo.OnStartDock Property

Occurs when the user begins to drag a control with a DragKind of dkDock.

```
property OnStartDock;
```

##### Description

Use the OnStartDock event handler to implement special processing when the user starts a drag-and-dock operation by dragging the control.

The OnStartDock event handler can create a TDragDockObjectEx object for the DragObject parameter to specify the appearance of the dragging rectangle and how the dragged control interacts with potential docking sites. If you return TDragDockObjectEx as the drag object, there is no need to call the Free method for the DragObject when dragging is over. If you use TDragDockObject, your application is responsible for freeing the drag object.

If the OnStartDock event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragDockObject object is automatically created.

#### 1.22.1.1.2.44 TComponentCombo.OnStartDrag Property

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

```
property OnStartDrag;
```

##### Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.



The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itse

#### 1.22.1.1.2.45 TComponentCombo.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

```
property ParentBiDiMode;
```

#### 1.22.1.1.2.46 TComponentCombo.ParentColor Property

Determines where a control looks for its color information.

```
property ParentColor;
```

#### 1.22.1.1.2.47 TComponentCombo.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

```
property ParentCtl3D;
```

##### Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

#### 1.22.1.1.2.48 TComponentCombo.ParentFont Property

Determines where a control looks for its font information.

```
property ParentFont;
```

#### 1.22.1.1.2.49 TComponentCombo.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

```
property ParentShowHint;
```

#### 1.22.1.1.2.50 TComponentCombo.PopupMenu Property

Identifies the pop-up menu associated with the control.

```
property PopupMenu;
```

##### Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

#### 1.22.1.1.2.51 TComponentCombo.ShowClassName Property

Specifies whether class names will be displayed in the items

```
property ShowClassName: Boolean;
```

**Description**

Default value is True.

**1.22.1.1.2.52 TComponentCombo.ShowComponents Property**

Determines if only TControl descendants will be shown in the list

```
property ShowComponents: Boolean;
```

**Description**

If ShowComponents is False, only controls (derived from TControl) will be shown in the list

Default value is True

**1.22.1.1.2.53 TComponentCombo.ShowHint Property**

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

**1.22.1.1.2.54 TComponentCombo.Sorted Property**

Determines whether the list portion of the combo box is alphabetized.

```
property Sorted;
```

**Description**

Set Sorted to true to sort the items in the Items list alphabetically. New items added to the list while Sorted is true are inserted in the correct alphabetical position.

When Sorted is changed from false to true, the original order of the items is lost. Setting Sorted back to false does not restore the original order.

**1.22.1.1.2.55 TComponentCombo.TabOrder Property**

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

**1.22.1.1.2.56 TComponentCombo.TabStop Property**

Determines if the user can tab to a control.

```
property TabStop;
```

**Description**

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

**1.22.1.1.2.57 TComponentCombo.Text Property**

Specifies the text string that is displayed in the edit box.

```
property Text;
```

**Description**

Use the Text property to read the text of the combo box or specify a new string for the Text value. By default, Text is the string specified in the Name property.

**Notes**

Setting Text to a string in the drop-down list causes that item to be selected. However, the value is considered a match only if it matches in a case-sensitive manner.

**1.22.1.1.2.58 TComponentCombo.Visible Property**

Determines whether the component appears on screen.

**property** Visible;

**Description**

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

**1.22.1.1.3 TComponentCombo Events****1.22.1.1.3.1 TComponentCombo.OnCanAddObject Event**

Occurs to defined whether object Obj can be added to drop-down list.

**property** OnCanAddObject: TCanAddObjectEvent;

**1.22.1.1.3.2 TComponentCombo.OnGetClassName Event**

Allows replacing displayed class name.

**property** OnGetClassName: TGetClassNameEvent;

**1.22.1.1.3.3 TComponentCombo.OnGetComponents Event**

Called during updating item list.

**property** OnGetComponents: TGetComponentsEvent;

**Description**

Write OnGetComponents handler to add additional items (objects) to combo-box. Additional object may not be owned by designed form.

**1.22.1.1.3.4 TComponentCombo.OnSelChanged Event**

Called when selected item (object) has bin changed by selecting from drop-down list.

**property** OnSelChanged: TSelChangedEvent;

**Description**

Write OnSelChanged handler to process selection of custom objects added using OnGetComponents (🔗 see page 358) event handler. If selected object is not owned by designed form Designer (🔗 see page 348) will not be able to select this object, so you should process selection manually.

---

**1.22.2 Types**

The following table lists types in this documentation.

Types

Type	Description
TCanAddObjectEvent (🔗 see page 359)	See TComponentCombo.OnCanAddObject (🔗 see page 358).
TGetClassNameEvent (🔗 see page 359)	See TComponentCombo.OnGetClassName Event (🔗 see page 358)
TGetComponentsEvent (🔗 see page 359)	See TComponentCombo.OnGetComponents (🔗 see page 358).
TSelChangedEvent (🔗 see page 359)	See TComponentCombo.OnSelChanged (🔗 see page 358)

1.22.2.1 edcCmbCombo.TCanAddObjectEvent Type

See TComponentCombo.OnCanAddObject (🔗 see page 358).

```
TCanAddObjectEvent = procedure (Sender: TObject; Obj: TPersistent; var CanAdd: Boolean) of
object;
```

**File**  
edcCmbCombo

1.22.2.2 edcCmbCombo.TGetClassNameEvent Type

See TComponentCombo.OnGetClassName Event (🔗 see page 358)

```
TGetClassNameEvent = procedure (Sender: TObject; Obj: TPersistent; var ClsName: string) of
object;
```

**File**  
edcCmbCombo

1.22.2.3 edcCmbCombo.TGetComponentsEvent Type

See TComponentCombo.OnGetComponents (🔗 see page 358).

```
TGetComponentsEvent = procedure (Sender: TObject; List: TList) of object;
```

**File**  
edcCmbCombo

1.22.2.4 edcCmbCombo.TSelChangedEvent Type

See TComponentCombo.OnSelChanged (🔗 see page 358)

```
TSelChangedEvent = procedure (Sender: TObject; SelObj: TPersistent) of object;
```

**File**  
edcCmbCombo

---

1.23 edcCompPal Namespace

## 1.23.1 Classes

The following table lists classes in this documentation.

### Classes

Class	Description
TPalettePanel (🔗 see page 360)	TPalettePanel is the panel to represent button panel with icons, where each icon represents installed component.
TPaletteTab (🔗 see page 377)	TPaletteTab is the container with tabs corresponding to component palette pages

### 1.23.1.1 TPalettePanel Class

TPalettePanel is the panel to represent button panel with icons, where each icon represents installed component.

#### Class Hierarchy



```
TPalettePanel = class(TCustomBtnPanel, IClassSelector);
```

#### File

edcCompPal

#### Description

This panel displays components of the specified Page (🔗 see page 376) with behavior similar to component' panel in the Delphi IDE.

#### Members

#### TCustomBtnPanel Methods





TCustomBtnPanel Methods	Description
🔗 ButtonAtPos (🔗 see page 22)	Returns the index of the button indicated by the coordinates of a point on the panel.
🔗 ButtonClick (🔗 see page 22)	Generates an OnButtonClick (🔗 see page 27) event.
🔗 ButtonRect (🔗 see page 22)	Indicates the rectangle occupied by the particular button.
🔗 CanAutoSize (🔗 see page 23)	Specifies whether the TCustomBtnPanel (🔗 see page 20) sizes itself automatically to accommodate its contents.
🔗 Create (🔗 see page 23)	Creates and initializes a TCustomBtnPanel instance.
🔗 Destroy (🔗 see page 23)	Destroys an instance of TCustomBtnPanel.
🔗 DrawButton (🔗 see page 23)	Generates an OnDrawButton (🔗 see page 27) event.
🔗 GetButtonHint (🔗 see page 24)	Returns hint text for the particular button.
🔗 InvalidateButtons (🔗 see page 24)	Repaints just pushed and released buttons.
🔗 Loaded (🔗 see page 24)	Finally initializes the TCustomBtnPanel (🔗 see page 20) after it is loaded from a stream.
🔗 MouseDown (🔗 see page 24)	Calls ButtonClick (🔗 see page 22) method.
🔗 Paint (🔗 see page 24)	Renders the image of a TCustomBtnPanel (🔗 see page 20).

#### IClassSelector Interface













IClassSelector Interface	Description
🔗 ClsChanged (🔗 see page 511)	Called when selected in component palette component class was changed.
🔗 ClsPalChanged (🔗 see page 512)	Called when component palette was changed.

#### TPalettePanel Class





























TPalettePanel Class	Description
🔗 ButtonClick (🔗 see page 364)	Generates an OnButtonClick event.
🔗 Create (🔗 see page 365)	Creates and initializes a TPalettePanel instance.

 Destroy (see page 365)	Destroys an instance of TPalettePanel.
 DrawButton (see page 365)	Generates an OnDrawButton event.
 GetButtonHint (see page 365)	Returns hint text for the particular button.
 UpdateList (see page 366)	Updates icons on the palette component

### TCustomBtnPanel Properties




TCustomBtnPanel Properties	Description
 AutoSize (see page 25)	Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 ButtonCount (see page 25)	Indicates the number of buttons in the array.
 ButtonHeight (see page 25)	Specifies the height of the buttons in array.
 ButtonWidth (see page 25)	Specifies the width of the buttons in array.
 Caption (see page 25)	Specifies a text string that identifies the control to the user. This is inherited property from TControl.
 DownButton (see page 26)	Specifies index of pressed button.
 Flat (see page 26)	Dictates whether the button should have a 2D look instead of the usual 3D look.
 HintProps (see page 26)	Provide properties to adjust hints processing.
 Margins (see page 26)	Specifies positioning of button array
 Orientation (see page 26)	Specifies whether the panel's array of button is horizontal or vertical.
 RowCount (see page 27)	Indicates the number of button rows in the panel.
 Transparent (see page 27)	Specifies whether the background of the button is transparent.

### TPalettePanel Class






TPalettePanel Class	Description
 Align (see page 366)	Determines how the control aligns within its container (parent control).
 Anchors (see page 366)	Specifies how the control is anchored to its parent.
 AutoSize (see page 367)	Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 BevelInner (see page 367)	Specifies the cut of the inner bevel.
 BevelOuter (see page 367)	Specifies the cut of the outer bevel.
 BevelWidth (see page 367)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
 BiDiMode (see page 368)	Specifies the bi-directional mode for the control.
 BorderStyle (see page 368)	Determines the style of the line drawn around the perimeter of the panel control.
 BorderWidth (see page 368)	Specifies the distance, in pixels, between the outer and inner bevels.
 ButtonHeight (see page 368)	Specifies the height of the buttons in array.
 ButtonWidth (see page 369)	Specifies the width of the buttons in array.
 Color (see page 369)	Specifies the background color of the control.
 Constraints (see page 369)	Specifies the size constraints for the control.
 Ctl3D (see page 369)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DownButton (see page 370)	Specifies index of pressed button.
 DragCursor (see page 370)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragImageType (see page 370)	Specifies drag image when dragging component on form.
 DragKind (see page 370)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 370)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 370)	Controls whether the control responds to mouse, keyboard, and timer events.
 Flat (see page 371)	Dictates whether the button should have a 2D look instead of the usual 3D look.
 Font (see page 371)	Controls the attributes of text written on or in the control.
 HintProps (see page 371)	Provide properties to adjust hints processing.
 Margins (see page 371)	Specifies positioning of button array
 OnButtonClick (see page 371)	Occurs when the user clicks the button on the underlying panel.
 OnCanResize (see page 372)	Occurs when an attempt is made to resize the control.
 OnClick (see page 372)	Occurs when the user clicks the control.
 OnConstrainedResize (see page 372)	Adjust resize constraints.
 OnContextPopup (see page 372)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

 OnDbClick (see page 373)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 373)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 373)	Occurs when the user drags an object over a control.
 OnEndDrag (see page 374)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 374)	Occurs when a control receives the input focus.
 OnExit (see page 374)	Occurs when the input focus shifts away from one control to another.
 OnMouseDown (see page 374)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 374)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 375)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnResize (see page 375)	Occurs immediately after the control is resized.
 OnStartDrag (see page 375)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 Orientation (see page 375)	Specifies whether the panel's array of button is horizontal or vertical.
 Page (see page 376)	Component's palette page name
 ParentBiDiMode (see page 376)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 376)	Determines where a control looks for its color information.
 ParentCtl3D (see page 376)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 376)	Determines where a control looks for its font information.
 ParentShowHint (see page 376)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 376)	Identifies the pop-up menu associated with the control.
 RowCount (see page 376)	Indicates the number of button rows in the panel.
 ShowHint (see page 377)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 TabOrder (see page 377)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 377)	Determines if the user can tab to a control.
 Transparent (see page 377)	Specifies whether the background of the button is transparent.
 Visible (see page 377)	Determines whether the component appears on screen.




## TCustomBtnPanel Events

TCustomBtnPanel Events	Description
 OnButtonClick (see page 27)	Occurs when the user clicks the button on the underlying panel.
 OnDrawButton (see page 27)	Occurs when a particular button on the panel needs to be drawn.
 OnGetButtonHint (see page 28)	Occurs before hint window will be displayed.






## Legend















	Method
	protected
	virtual
	Property
	Event

## TCustomBtnPanel Events





TCustomBtnPanel Events	Description
 OnButtonClick (see page 27)	Occurs when the user clicks the button on the underlying panel.
 OnDrawButton (see page 27)	Occurs when a particular button on the panel needs to be drawn.
 OnGetButtonHint (see page 28)	Occurs before hint window will be displayed.

## TCustomBtnPanel Methods













TCustomBtnPanel Methods	Description
 ButtonAtPos (see page 22)	Returns the index of the button indicated by the coordinates of a point on the panel.
 ButtonClick (see page 22)	Generates an OnButtonClick (see page 27) event.
 ButtonRect (see page 22)	Indicates the rectangle occupied by the particular button.
 CanAutoSize (see page 23)	Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents.
 Create (see page 23)	Creates and initializes a TCustomBtnPanel instance.

  Destroy (see page 23)	Destroys an instance of TCustomBtnPanel.
  DrawButton (see page 23)	Generates an OnDrawButton (see page 27) event.
  GetButtonHint (see page 24)	Returns hint text for the particular button.
  InvalidateButtons (see page 24)	Repaints just pushed and released buttons.
  Loaded (see page 24)	Finally initializes the TCustomBtnPanel (see page 20) after it is loaded from a stream.
  MouseDown (see page 24)	Calls ButtonClick (see page 22) method.
  Paint (see page 24)	Renders the image of a TCustomBtnPanel (see page 20).













### IClassSelector Interface

IClassSelector Interface	Description
  ClsChanged (see page 511)	Called when selected in component palette component class was changed.
  ClsPalChanged (see page 512)	Called when component palette was changed.















### TPalettePanel Class

TPalettePanel Class	Description
  ButtonClick (see page 364)	Generates an OnButtonClick event.
  Create (see page 365)	Creates and initializes a TPalettePanel instance.
  Destroy (see page 365)	Destroys an instance of TPalettePanel.
  DrawButton (see page 365)	Generates an OnDrawButton event.
  GetButtonHint (see page 365)	Returns hint text for the particular button.
  UpdateList (see page 366)	Updates icons on the palette component





























### TCustomBtnPanel Properties

TCustomBtnPanel Properties	Description
 AutoSize (see page 25)	Specifies whether the TCustomBtnPanel (see page 20) sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 ButtonCount (see page 25)	Indicates the number of buttons in the array.
 ButtonHeight (see page 25)	Specifies the height of the buttons in array.
 ButtonWidth (see page 25)	Specifies the width of the buttons in array.
 Caption (see page 25)	Specifies a text string that identifies the control to the user. This is inherited property from TControl.
 DownButton (see page 26)	Specifies index of pressed button.
 Flat (see page 26)	Dictates whether the button should have a 2D look instead of the usual 3D look.
 HintProps (see page 26)	Provide properties to adjust hints processing.
 Margins (see page 26)	Specifies positioning of button array
 Orientation (see page 26)	Specifies whether the panel's array of button is horizontal or vertical.
 RowCount (see page 27)	Indicates the number of button rows in the panel.
 Transparent (see page 27)	Specifies whether the background of the button is transparent.

### TPalettePanel Class

TPalettePanel Class	Description
 Align (see page 366)	Determines how the control aligns within its container (parent control).
 Anchors (see page 366)	Specifies how the control is anchored to its parent.
 AutoSize (see page 367)	Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents. This is inherited property from TControl.
 BevelInner (see page 367)	Specifies the cut of the inner bevel.
 BevelOuter (see page 367)	Specifies the cut of the outer bevel.
 BevelWidth (see page 367)	Determines the width, in pixels, of both the inner and outer bevels of a panel.
 BiDiMode (see page 368)	Specifies the bi-directional mode for the control.
 BorderStyle (see page 368)	Determines the style of the line drawn around the perimeter of the panel control.
 BorderWidth (see page 368)	Specifies the distance, in pixels, between the outer and inner bevels.
 ButtonHeight (see page 368)	Specifies the height of the buttons in array.
 ButtonWidth (see page 369)	Specifies the width of the buttons in array.
 Color (see page 369)	Specifies the background color of the control.
 Constraints (see page 369)	Specifies the size constraints for the control.
 Ctl3D (see page 369)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.



 DownButton (see page 370)	Specifies index of pressed button.
 DragCursor (see page 370)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragImageType (see page 370)	Specifies drag image when dragging component on form.
 DragKind (see page 370)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 370)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 370)	Controls whether the control responds to mouse, keyboard, and timer events.
 Flat (see page 371)	Dictates whether the button should have a 2D look instead of the usual 3D look.
 Font (see page 371)	Controls the attributes of text written on or in the control.
 HintProps (see page 371)	Provide properties to adjust hints processing.
 Margins (see page 371)	Specifies positioning of button array
 OnButtonClick (see page 371)	Occurs when the user clicks the button on the underlying panel.
 OnCanResize (see page 372)	Occurs when an attempt is made to resize the control.
 OnClick (see page 372)	Occurs when the user clicks the control.
 OnConstrainedResize (see page 372)	Adjust resize constraints.
 OnContextPopup (see page 372)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 373)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 373)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 373)	Occurs when the user drags an object over a control.
 OnEndDrag (see page 374)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 374)	Occurs when a control receives the input focus.
 OnExit (see page 374)	Occurs when the input focus shifts away from one control to another.
 OnMouseDown (see page 374)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 374)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 375)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnResize (see page 375)	Occurs immediately after the control is resized.
 OnStartDrag (see page 375)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 Orientation (see page 375)	Specifies whether the panel's array of button is horizontal or vertical.
 Page (see page 376)	Component's palette page name
 ParentBiDiMode (see page 376)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 376)	Determines where a control looks for its color information.
 ParentCtl3D (see page 376)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 376)	Determines where a control looks for its font information.
 ParentShowHint (see page 376)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 376)	Identifies the pop-up menu associated with the control.
 RowCount (see page 376)	Indicates the number of button rows in the panel.
 ShowHint (see page 377)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 TabOrder (see page 377)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 377)	Determines if the user can tab to a control.
 Transparent (see page 377)	Specifies whether the background of the button is transparent.
 Visible (see page 377)	Determines whether the component appears on screen.

### 1.23.1.1.1 TPalettePanel Methods

#### 1.23.1.1.1.1 TPalettePanel.ButtonClick Method

Generates an OnButtonClick event.

**procedure** ButtonClick(AButton: integer; Shift: TShiftState); **override**;

**Description**

ButtonClick is called automatically when user presses mouse button with the mouse pointer over the particular button. Then it generates an OnButtonClick event.

The AButton parameter indicates index of corresponding button.

The Shift parameter indicates which shift keys (Shift, Ctrl, or Alt) were down when the user pressed the mouse button.

Override this method to add class-specific processing when the button clicks.

**Example**

See (see page 10) how descendant of TCustomBtnPanel overrides this method to assign current class type in DsnManager

**1.23.1.1.1.2 TPalettePanel.Create Constructor**

Creates and initializes a TPalettePanel instance.

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

**Description**

Use Create to programmatically instantiate a TPalettePanel component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

**1.23.1.1.1.3 TPalettePanel.Destroy Destructor**

Destroys an instance of TPalettePanel.

```
destructor Destroy; override;
```

**Description**

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

**1.23.1.1.1.4 TPalettePanel.DrawButton Method**

Generates an OnDrawButton event.

```
procedure DrawButton(var ARect: TRect; Index: Integer); override;
```

**Description**

Simply generates an OnDrawButton event after rendering this button in TCustomBtnPanel.Paint.

The Rect parameter indicates the location of the button on the canvas.

The Index parameter indicates index of corresponding button.

**1.23.1.1.1.5 TPalettePanel.GetButtonHint Method**

Returns hint text for the particular button.

```
function GetButtonHint(Index: Integer): WideString; override;
```

**Description**

Returns hint text for the particular button.

The Index parameter indicates index of corresponding button.

#### 1.23.1.1.1.6 TPalettePanel.UpdateList Method

Updates icons on the palette component

```
procedure UpdateList;
```

##### Description

This method checks PackageMng for corresponding set of components and updates palette component's set of icons.

#### 1.23.1.1.2 TPalettePanel Properties

##### 1.23.1.1.2.1 TPalettePanel.Align Property

Determines how the control aligns within its container (parent control).

```
property Align;
```

##### Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to `alLeft`. The value of `alLeft` for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is `alNone`, which means a control remains where it is positioned on a form or panel.

**Tip:** If Align is set to `alClient`, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

**Note:** To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

##### 1.23.1.1.2.2 TPalettePanel.Anchors Property

Specifies how the control is anchored to its parent.

```
property Anchors;
```

##### Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to `[akLeft, akRight]`, the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

**Note:** If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

#### 1.23.1.1.2.3 TPalettePanel.AutoSize Property

Specifies whether the TCustomBtnPanel sizes itself automatically to accommodate its contents. This is inherited property from TControl.

```
property AutoSize;
```

##### Description

Use AutoSize to specify whether the TCustomBtnPanel sizes itself automatically. When AutoSize is True, the TCustomBtnPanel resizes automatically to arrange all the buttons in array.

By default, AutoSize is True.

#### 1.23.1.1.2.4 TPalettePanel.BevelInner Property

Specifies the cut of the inner bevel.

```
property BevelInner;
```

##### Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

#### 1.23.1.1.2.5 TPalettePanel.BevelOuter Property

Specifies the cut of the outer bevel.

```
property BevelOuter;
```

##### Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

#### 1.23.1.1.2.6 TPalettePanel.BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

```
property BevelWidth;
```

##### Description

Use BevelWidth to specify how wide the inner or outer bevel should be. Do not confuse BevelWidth, which is the width of the bevels, with BorderWidth, which is the space between the bevels.

If both the BevelInner and BevelOuter properties are bvNone, BevelWidth has no effect. To remove both bevels, set the

BevellInner and BevelOuter properties to bvNone, rather than setting the BevelWidth to 0, as this involves less overhead when painting.

#### 1.23.1.1.2.7 TPalettePanel.BiDiMode Property

Specifies the bi-directional mode for the control.

**property** BiDiMode;

##### Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

#### 1.23.1.1.2.8 TPalettePanel.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

**property** BorderStyle;

##### Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

#### 1.23.1.1.2.9 TPalettePanel.BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

**property** BorderWidth;

##### Description

Use BorderWidth to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

The border of a panel is the area between the outer and inner bevels. It is visible only if the inner bevel is raised or lowered, but affects the inset of the caption within the panel even if BevellInner is bvNone. If the Alignment property is not taCenter, the Caption will be aligned to the inner edge of the border. This edge is BorderWidth pixels in from the outer bevel if BevellInner is bvNone. It is the inner edge of the inner bevel otherwise.

Do not confuse the border of the panel with line drawn around the panel itself. The line around the panel is specified by the BorderStyle property.

#### 1.23.1.1.2.10 TPalettePanel.ButtonHeight Property

Specifies the height of the buttons in array.

```
property ButtonHeight: integer;
```

**Description**

ButtonHeight represents the height, in pixels, of the buttons on the panel.

Setting this property to value less than 2 pixels throws an Exception.

By default, ButtonHeight is 25.

**1.23.1.1.2.11 TPalettePanel.ButtonWidth Property**

Specifies the width of the buttons in array.

```
property ButtonWidth: integer;
```

**Description**

ButtonWidth represents the width, in pixels, of the buttons on the panel.

Setting this property to value less than 2 pixels throws an Exception.

By default, ButtonWidth is 25.

**1.23.1.1.2.12 TPalettePanel.Color Property**

Specifies the background color of the control.

```
property Color;
```

**Description**

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

**1.23.1.1.2.13 TPalettePanel.Constraints Property**

Specifies the size constraints for the control.

```
property Constraints;
```

**Description**

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

**Warning:** Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

**1.23.1.1.2.14 TPalettePanel.Ctl3D Property**

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

```
property Ctl3D;
```

**Description**

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

### 1.23.1.1.2.15 TPalettePanel.DownButton Property

Specifies index of pressed button.

```
property DownButton: integer;
```

#### Description

Specifies index of the button which is in selected state where 0 is the first button, 1 is the second and so on;

Set this property to change selected button in the array.

By default, DownButton is -1;

### 1.23.1.1.2.16 TPalettePanel.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

```
property DragCursor;
```

#### Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

**Note:** To make a custom cursor available for the DragCursor property, see the Cursor property.

### 1.23.1.1.2.17 TPalettePanel.DragImageType Property

Specifies drag image when dragging component on form.

```
property DragImageType: TComponentClassDragImage;
```

#### Remarks

In Delphi 5,6,7 you will need to add csDisplayDragImage to designed form's ControlStyle.

In Delphi 2005,2006,2007,2009 - DragObject.AlwaysShowDragImages := True, so dragged image is shown over any control.

### 1.23.1.1.2.18 TPalettePanel.DragKind Property

Specifies whether the control is being dragged normally or for docking.

```
property DragKind;
```

#### Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

### 1.23.1.1.2.19 TPalettePanel.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

```
property DragMode;
```

#### Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

### 1.23.1.1.2.20 TPalettePanel.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

```
property Enabled;
```

**Description**

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

**1.23.1.1.2.21 TPalettePanel.Flat Property**

Dictates whether the button should have a 2D look instead of the usual 3D look.

```
property Flat: Boolean;
```

**Description**

Set Flat to True if you want the button to display the button without the edge bevel that gives buttons a 3D look.

By default, Flat is True.

**1.23.1.1.2.22 TPalettePanel.Font Property**

Controls the attributes of text written on or in the control.

```
property Font;
```

**Description**

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

**1.23.1.1.2.23 TPalettePanel.HintProps Property**

Provide properties to adjust hints processing.

```
property HintProps: TechHintHelper;
```

**1.23.1.1.2.24 TPalettePanel.Margins Property**

Specifies positioning of button array

```
property Margins: TBtnMargins;
```

**Description**

Margins specifies the Left, Top, Right and Bottom margins between buttons and underlying panel as well as horizontal and vertical spaces between buttons themselves.

**Example**

This example (see page 10) demonstrates how to set Margins property at run-time

**1.23.1.1.2.25 TPalettePanel.OnButtonClick Property**

Occurs when the user clicks the button on the underlying panel.

```
property OnButtonClick: TButtonClickEvent;
```

**Description**

Use the OnButtonClick event handler to respond when the user clicks the button on the TCustomBtnPanel.

OnButtonClick occurs when the user presses mouse button with the mouse pointer over the corresponding rectangle of the underlying TCustomBtnPanel.



The Sender parameter is the object whose event handler is called.

The Index parameter indicates index of pressed button.

#### 1.23.1.1.2.26 TPalettePanel.OnCanResize Property

Occurs when an attempt is made to resize the control.

```
property OnCanResize;
```

##### Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

#### 1.23.1.1.2.27 TPalettePanel.OnClick Property

Occurs when the user clicks the control.

```
property OnClick;
```

#### 1.23.1.1.2.28 TPalettePanel.OnConstrainedResize Property

Adjust resize constraints.

```
property OnConstrainedResize;
```

##### Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

##### Notes

The OnConstrainedResize handler is called immediately after the OnCanResize handler.

#### 1.23.1.1.2.29 TPalettePanel.OnContextPopup Property

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

```
property OnContextPopup;
```

##### Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM\_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also

be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

**Note:** Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

### 1.23.1.1.2.30 TPalettePanel.OnDbClick Property

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

**property** OnDbClick;

#### Description

Use the OnDbClick event to respond to mouse double-clicks.

### 1.23.1.1.2.31 TPalettePanel.OnDragDrop Property

Occurs when the user drops an object being dragged.

**property** OnDragDrop;

#### Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

### 1.23.1.1.2.32 TPalettePanel.OnDragOver Property

Occurs when the user drags an object over a control.

**property** OnDragOver;

#### Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

**Note:** Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

### 1.23.1.1.2.33 TPalettePanel.OnEndDrag Property

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

```
property OnEndDrag;
```

#### Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

### 1.23.1.1.2.34 TPalettePanel.OnEnter Property

Occurs when a control receives the input focus.

```
property OnEnter;
```

#### Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

### 1.23.1.1.2.35 TPalettePanel.OnExit Property

Occurs when the input focus shifts away from one control to another.

```
property OnExit;
```

#### Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

### 1.23.1.1.2.36 TPalettePanel.OnMouseDown Property

Occurs when the user presses a mouse button with the mouse pointer over a control.

```
property OnMouseDown;
```

#### Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

### 1.23.1.1.2.37 TPalettePanel.OnMouseMove Property

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

```
property OnMouseMove;
```

#### Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

### 1.23.1.1.2.38 TPalettePanel.OnMouseUp Property

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

```
property OnMouseUp;
```

#### Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

### 1.23.1.1.2.39 TPalettePanel.OnResize Property

Occurs immediately after the control is resized.

```
property OnResize;
```

#### Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

### 1.23.1.1.2.40 TPalettePanel.OnStartDrag Property

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

```
property OnStartDrag;
```

#### Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itse

### 1.23.1.1.2.41 TPalettePanel.Orientation Property

Specifies whether the panel's array of button is horizontal or vertical.

```
property Orientation: TRowOrientation;
```

#### Description

Use Orientation to specify whether the the panel's array of button is horizontal or vertical.

By default, Orientation is roHorizontal.

#### 1.23.1.1.2.42 TPalettePanel.Page Property

Component's palette page name

```
property Page: string;
```

##### Description

Specifies name of the page with components icons.

Set this property to change this name.

#### 1.23.1.1.2.43 TPalettePanel.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

```
property ParentBiDiMode;
```

#### 1.23.1.1.2.44 TPalettePanel.ParentColor Property

Determines where a control looks for its color information.

```
property ParentColor;
```

#### 1.23.1.1.2.45 TPalettePanel.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

```
property ParentCtl3D;
```

##### Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

#### 1.23.1.1.2.46 TPalettePanel.ParentFont Property

Determines where a control looks for its font information.

```
property ParentFont;
```

#### 1.23.1.1.2.47 TPalettePanel.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

```
property ParentShowHint;
```

#### 1.23.1.1.2.48 TPalettePanel.PopupMenu Property

Identifies the pop-up menu associated with the control.

```
property PopupMenu;
```

##### Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

#### 1.23.1.1.2.49 TPalettePanel.RowCount Property

Indicates the number of button rows in the panel.

```
property RowCount: integer;
```

**Description**

Read RowCount to determine the number of rows in the button's array are placed.

Set RowCount to add or delete rows to rearrange buttons.

By default, RowCount is 1.

**1.23.1.1.2.50 TPalettePanel.ShowHint Property**

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

**1.23.1.1.2.51 TPalettePanel.TabOrder Property**

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

**1.23.1.1.2.52 TPalettePanel.TabStop Property**

Determines if the user can tab to a control.

```
property TabStop;
```

**Description**

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

**1.23.1.1.2.53 TPalettePanel.Transparent Property**

Specifies whether the background of the button is transparent.

```
property Transparent: Boolean;
```

**Description**

Use Transparent to specify whether the background of the button is transparent.

**1.23.1.1.2.54 TPalettePanel.Visible Property**

Determines whether the component appears on screen.

```
property Visible;
```

**Description**

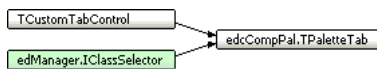
Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

**1.23.1.2 TPaletteTab Class**

TPaletteTab is the container with tabs corresponding to component palette pages

## Class Hierarchy



```
TPaletteTab = class(TCustomTabControl, IClassSelector);
```

## File

edcCompPal

## Description

TPaletteTab provides look and feel of a TPalette component similar to standard Borland IDE one.

With TPaletteTab component all the installed components set themselves up to tabs with group names.

User can shift between component groups clicking on TPaletteTab tabs.

## Members

### IClassSelector Methods

IClassSelector Methods	Description
✦ ClsChanged (see page 511)	Called when selected in component palette component class was changed.
✦ ClsPalChanged (see page 512)	Called when component palette was changed.




### TPaletteTab Properties

#### TPaletteTab Class



TPaletteTab Class	Description
Align (see page 380)	Determines how the control aligns within its container (parent control).
Anchors (see page 380)	Specifies how the control is anchored to its parent.
BiDiMode (see page 381)	Specifies the bi-directional mode for the control.
Constraints (see page 381)	Specifies the size constraints for the control.
DragCursor (see page 381)	Indicates the image used to represent the mouse pointer when the control is being dragged.
DragKind (see page 381)	Specifies whether the control is being dragged normally or for docking.
DragMode (see page 381)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
Enabled (see page 381)	Controls whether the control responds to mouse, keyboard, and timer events.
Flat (see page 382)	Determines whether the button within component palettes has a 3D border that provides a raised or lowered look.
Font (see page 382)	Controls the attributes of text written on or in the control.
HintProps (see page 382)	Provide properties to adjust hints processing.
HotTrack (see page 382)	Determines whether labels on the tab under the mouse are automatically highlighted.
Images (see page 382)	Specifies the images drawn in tabs.
MultiLine (see page 382)	Determines whether the tabs can appear on more than one row.
OnChange (see page 383)	Occurs after a new tab is selected.
OnChanging (see page 383)	Occurs immediately before a new tab is selected.
OnContextPopup (see page 383)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
OnDragDrop (see page 383)	Occurs when the user drops an object being dragged.
OnDragOver (see page 383)	Occurs when the user drags an object over a control.
OnDrawTab (see page 383)	Occurs when a tab is about to be drawn.
OnEndDock (see page 384)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
OnEndDrag (see page 384)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
OnEnter (see page 384)	Occurs when a control receives the input focus.
OnExit (see page 384)	Occurs when the input focus shifts away from one control to another.
OnGetImageIndex (see page 384)	Occurs when a tab is about to display its associated image.
OnMouseDown (see page 384)	Occurs when the user presses a mouse button with the mouse pointer over a control.

 OnMouseMove (see page 385)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control
 OnMouseUp (see page 385)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnResize (see page 385)	Occurs immediately after the control is resized.
 OnStartDrag (see page 385)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 OwnerDraw (see page 385)	Specifies whether the tab control handles its own painting.
  PalettePanel (see page 386)	Palette panel with components of the page, selected in palette tab.
 ParentBiDiMode (see page 386)	Specifies whether the control uses its parent's BiDiMode (see page 381).
 ParentFont (see page 386)	Determines where a control looks for its font information.
 ParentShowHint (see page 386)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 386)	Identifies the pop-up menu associated with the control.
 RaggedRight (see page 386)	Specifies whether rows of tabs stretch to fill the width of the control.
 ResetOnChange (see page 386)	Specifies if current selected component class will be reset after changing tab page
 ScrollOpposite (see page 387)	Determines how the rows of tabs are scrolled in a multiline tab control.
 ShowHint (see page 387)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 Style (see page 387)	Specifies the style of the tab control.
 TabHeight (see page 387)	Specifies the height, in pixels, of the tabs in the tab control.
 TabIndex (see page 387)	Identifies the selected tab on a tab control.
 TabOrder (see page 387)	Indicates the position of the control in its parent's tab order.
 TabPosition (see page 388)	Determines whether tabs appear at the top or bottom.
 Tabs (see page 388)	Contains the list of text strings that label the tabs of the tab control.
 TabStop (see page 388)	Determines if the user can tab to a control.
 TabWidth (see page 388)	Specifies the horizontal size, in pixels, of the tabs in the tab control.
 Visible (see page 388)	Determines whether the component appears onscreen.

## Legend
















	Method
	Property
	read only

## IClassSelector Methods

IClassSelector Methods	Description
 ClsChanged (see page 511)	Called when selected in component palette component class was changed.
 ClsPalChanged (see page 512)	Called when component palette was changed.

## TPaletteTab Properties

### TPaletteTab Class

TPaletteTab Class	Description
 Align (see page 380)	Determines how the control aligns within its container (parent control).
 Anchors (see page 380)	Specifies how the control is anchored to its parent.
 BiDiMode (see page 381)	Specifies the bi-directional mode for the control.
 Constraints (see page 381)	Specifies the size constraints for the control.
 DragCursor (see page 381)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 381)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 381)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 381)	Controls whether the control responds to mouse, keyboard, and timer events.
 Flat (see page 382)	Determines whether the button within component palettes has a 3D border that provides a raised or lowered look.
 Font (see page 382)	Controls the attributes of text written on or in the control.
 HintProps (see page 382)	Provide properties to adjust hints processing.
 HotTrack (see page 382)	Determines whether labels on the tab under the mouse are automatically highlighted.
 Images (see page 382)	Specifies the images drawn in tabs.
 MultiLine (see page 382)	Determines whether the tabs can appear on more than one row.
 OnChange (see page 383)	Occurs after a new tab is selected.



 OnChanging (see page 383)	Occurs immediately before a new tab is selected.
 OnContextPopup (see page 383)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDragDrop (see page 383)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 383)	Occurs when the user drags an object over a control.
 OnDrawTab (see page 383)	Occurs when a tab is about to be drawn.
 OnEndDock (see page 384)	Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.
 OnEndDrag (see page 384)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 384)	Occurs when a control receives the input focus.
 OnExit (see page 384)	Occurs when the input focus shifts away from one control to another.
 OnGetImageIndex (see page 384)	Occurs when a tab is about to display its associated image.
 OnMouseDown (see page 384)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 385)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 385)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnResize (see page 385)	Occurs immediately after the control is resized.
 OnStartDrag (see page 385)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 OwnerDraw (see page 385)	Specifies whether the tab control handles its own painting.
 PalettePanel (see page 386)	Palette panel with components of the page, selected in palette tab.
 ParentBiDiMode (see page 386)	Specifies whether the control uses its parent's BiDiMode (see page 381).
 ParentFont (see page 386)	Determines where a control looks for its font information.
 ParentShowHint (see page 386)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 386)	Identifies the pop-up menu associated with the control.
 RaggedRight (see page 386)	Specifies whether rows of tabs stretch to fill the width of the control.
 ResetOnChange (see page 386)	Specifies if current selected component class will be reset after changing tab page.
 ScrollOpposite (see page 387)	Determines how the rows of tabs are scrolled in a multiline tab control.
 ShowHint (see page 387)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 Style (see page 387)	Specifies the style of the tab control.
 TabHeight (see page 387)	Specifies the height, in pixels, of the tabs in the tab control.
 TabIndex (see page 387)	Identifies the selected tab on a tab control.
 TabOrder (see page 387)	Indicates the position of the control in its parent's tab order.
 TabPosition (see page 388)	Determines whether tabs appear at the top or bottom.
 Tabs (see page 388)	Contains the list of text strings that label the tabs of the tab control.
 TabStop (see page 388)	Determines if the user can tab to a control.
 TabWidth (see page 388)	Specifies the horizontal size, in pixels, of the tabs in the tab control.
 Visible (see page 388)	Determines whether the component appears onscreen.

## 1.23.1.2.1 TPaletteTab Properties

### 1.23.1.2.1.1 TPaletteTab.Align Property

Determines how the control aligns within its container (parent control).

**property** Align;

#### Description

This is inherited property.

See TControl.Align for details

### 1.23.1.2.1.2 TPaletteTab.Anchors Property

Specifies how the control is anchored to its parent.

**property** Anchors;

**Description**

This is inherited property.

See TControl.Anchors for details

**1.23.1.2.1.3 TPaletteTab.BiDiMode Property**

Specifies the bi-directional mode for the control.

```
property BiDiMode;
```

**Description**

This is inherited property.

See TControl.BiDiMode for details

**1.23.1.2.1.4 TPaletteTab.Constraints Property**

Specifies the size constraints for the control.

```
property Constraints;
```

**Description**

This is inherited property.

See TControl.Constraints for details

**1.23.1.2.1.5 TPaletteTab.DragCursor Property**

Indicates the image used to represent the mouse pointer when the control is being dragged.

```
property DragCursor;
```

**Description**

This is inherited property.

See TControl.DragCursor for details

**1.23.1.2.1.6 TPaletteTab.DragKind Property**

Specifies whether the control is being dragged normally or for docking.

```
property DragKind;
```

**Description**

This is inherited property.

See TControl.DragKind for details

**1.23.1.2.1.7 TPaletteTab.DragMode Property**

Determines how the control initiates drag-and-drop or drag-and-dock operations.

```
property DragMode;
```

**Description**

This is inherited property.

See TControl.DragMode for details

**1.23.1.2.1.8 TPaletteTab.Enabled Property**

Controls whether the control responds to mouse, keyboard, and timer events.

**property** Enabled;

#### Description

This is inherited property.

See TControl.Enabled for details

### 1.23.1.2.1.9 TPaletteTab.Flat Property

Determines whether the button within component palettes has a 3D border that provides a raised or lowered look.

**property** Flat: Boolean;

#### Description

Set Flat to True to remove the raised border when the button is unselected and the lowered border when the button is clicked or selected.

### 1.23.1.2.1.10 TPaletteTab.Font Property

Controls the attributes of text written on or in the control.

**property** Font;

#### Description

This is inherited property.

See TControl.Font for details

### 1.23.1.2.1.11 TPaletteTab.HintProps Property

Provide properties to adjust hints processing.

**property** HintProps: TechHintHelper;

### 1.23.1.2.1.12 TPaletteTab.HotTrack Property

Determines whether labels on the tab under the mouse are automatically highlighted.

**property** HotTrack;

#### Description

This is inherited property.

See TCustomTabControl.HotTrack for details

### 1.23.1.2.1.13 TPaletteTab.Images Property

Specifies the images drawn in tabs.

**property** Images;

#### Description

This is inherited property.

See TCustomTabControl.Images for details

### 1.23.1.2.1.14 TPaletteTab.MultiLine Property

Determines whether the tabs can appear on more than one row.

**property** MultiLine;

#### Description

This is inherited property.

See TCustomTabControl.MultiLine for details

#### 1.23.1.2.1.15 TPaletteTab.OnChange Property

Occurs after a new tab is selected.

**property** OnChange;

##### Description

This is inherited property.

See TCustomTabControl.OnChange for details

#### 1.23.1.2.1.16 TPaletteTab.OnChanging Property

Occurs immediately before a new tab is selected.

**property** OnChanging;

##### Description

This is inherited property.

See TCustomTabControl.OnChanging for details

#### 1.23.1.2.1.17 TPaletteTab.OnContextPopup Property

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

**property** OnContextPopup;

##### Description

This is inherited property.

See TControl.OnContextPopup for details

#### 1.23.1.2.1.18 TPaletteTab.OnDragDrop Property

Occurs when the user drops an object being dragged.

**property** OnDragDrop;

##### Description

This is inherited property.

See TControl.OnDragDrop for details

#### 1.23.1.2.1.19 TPaletteTab.OnDragOver Property

Occurs when the user drags an object over a control.

**property** OnDragOver;

##### Description

This is inherited property.

See TControl.OnDragOver for details

#### 1.23.1.2.1.20 TPaletteTab.OnDrawTab Property

Occurs when a tab is about to be drawn.

**property** OnDrawTab;

**Description**

This is inherited property.

See TCustomTabControl.OnDrawTab for details

**1.23.1.2.1.21 TPaletteTab.OnEndDock Property**

Occurs when the dragging of an object ends, either by docking the object or by canceling the dragging.

**property** OnEndDock;

**Description**

This is inherited property.

See TControl.OnEndDock for details

**1.23.1.2.1.22 TPaletteTab.OnEndDrag Property**

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

**property** OnEndDrag;

**Description**

This is inherited property.

See TControl.OnEndDrag for details

**1.23.1.2.1.23 TPaletteTab.OnEnter Property**

Occurs when a control receives the input focus.

**property** OnEnter;

**Description**

This is inherited property.

See TWinControl.OnEnter for details

**1.23.1.2.1.24 TPaletteTab.OnExit Property**

Occurs when the input focus shifts away from one control to another.

**property** OnExit;

**Description**

This is inherited property.

See TWinControl.OnExit for details

**1.23.1.2.1.25 TPaletteTab.OnGetImageIndex Property**

Occurs when a tab is about to display its associated image.

**property** OnGetImageIndex;

**Description**

This is inherited property.

See TCustomTabControl.OnGetImageIndex for details

**1.23.1.2.1.26 TPaletteTab.OnMouseDown Property**

Occurs when the user presses a mouse button with the mouse pointer over a control.

**property** OnMouseDown;

#### Description

This is inherited property.

See TControl.OnMouseDown for details

### 1.23.1.2.1.27 TPaletteTab.OnMouseMove Property

Occurs when the user moves the mouse pointer while the mouse pointer is over a control

**property** OnMouseMove;

#### Description

This is inherited property.

See TControl.OnMouseMove for details

### 1.23.1.2.1.28 TPaletteTab.OnMouseUp Property

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

**property** OnMouseUp;

#### Description

This is inherited property.

See TControl.OnMouseUp for details

### 1.23.1.2.1.29 TPaletteTab.OnResize Property

Occurs immediately after the control is resized.

**property** OnResize;

#### Description

This is inherited property.

See TControl.OnResize for details

### 1.23.1.2.1.30 TPaletteTab.OnStartDrag Property

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

**property** OnStartDrag;

#### Description

This is inherited property.

See TControl.OnStartDrag for details

### 1.23.1.2.1.31 TPaletteTab.OwnerDraw Property

Specifies whether the tab control handles its own painting.

**property** OwnerDraw;

#### Description

This is inherited property.

See TCustomTabControl.OwnerDraw for details

### 1.23.1.2.1.32 TPaletteTab.PalettePanel Property

Palette panel with components of the page, selected in palette tab.

```
property PalettePanel: TPalettePanel;
```

### 1.23.1.2.1.33 TPaletteTab.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode (see page 381).

```
property ParentBiDiMode;
```

#### Description

This is inherited property.

See TControl.ParentBiDiMode for details

### 1.23.1.2.1.34 TPaletteTab.ParentFont Property

Determines where a control looks for its font information.

```
property ParentFont;
```

#### Description

This is inherited property.

See TControl.ParentFont for details

### 1.23.1.2.1.35 TPaletteTab.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

```
property ParentShowHint;
```

#### Description

This is inherited property.

See TControl.ParentShowHint for details

### 1.23.1.2.1.36 TPaletteTab.PopupMenu Property

Identifies the pop-up menu associated with the control.

```
property PopupMenu;
```

#### Description

This is inherited property.

See TControl.PopupMenu for details

### 1.23.1.2.1.37 TPaletteTab.RaggedRight Property

Specifies whether rows of tabs stretch to fill the width of the control.

```
property RaggedRight;
```

#### Description

This is inherited property.

See TCustomTabControl.RaggedRight for details

### 1.23.1.2.1.38 TPaletteTab.ResetOnChange Property

Specifies if current selected component class will be reset after changing tab page

```
property ResetOnChange: Boolean;
```

**Description****1.23.1.2.1.39 TPaletteTab.ScrollOpposite Property**

Determines how the rows of tabs are scrolled in a multiline tab control.

```
property ScrollOpposite;
```

**Description**

This is inherited property.

See TCustomTabControl.ScrollOpposite for details

**1.23.1.2.1.40 TPaletteTab.ShowHint Property**

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

**Description**

This is inherited property.

See TControl.ShowHint for details

**1.23.1.2.1.41 TPaletteTab.Style Property**

Specifies the style of the tab control.

```
property Style;
```

**Description**

This is inherited property.

See TCustomTabControl.Style for details

**1.23.1.2.1.42 TPaletteTab.TabHeight Property**

Specifies the height, in pixels, of the tabs in the tab control.

```
property TabHeight;
```

**Description**

This is inherited property.

See TCustomTabControl.TabHeight for details

**1.23.1.2.1.43 TPaletteTab.TabIndex Property**

Identifies the selected tab on a tab control.

```
property TabIndex;
```

**Description**

This is inherited property.

See TCustomTabControl.TabIndex for details

**1.23.1.2.1.44 TPaletteTab.TabOrder Property**

Indicates the position of the control in its parent's tab order.



```
property TabOrder;
```

**Description**

This is inherited property.

See TWinControl.TabOrder for details

**1.23.1.2.1.45 TPaletteTab.TabPosition Property**

Determines whether tabs appear at the top or bottom.

```
property TabPosition;
```

**Description**

This is inherited property.

See TCustomTabControl.TabPosition for details

**1.23.1.2.1.46 TPaletteTab.Tabs Property**

Contains the list of text strings that label the tabs of the tab control.

```
property Tabs;
```

**Description**

This is inherited property.

See TCustomTabControl.Tabs for details

**1.23.1.2.1.47 TPaletteTab.TabStop Property**

Determines if the user can tab to a control.

```
property TabStop;
```

**Description**

This is inherited property.

See TWinControl.TabStop for details

**1.23.1.2.1.48 TPaletteTab.TabWidth Property**

Specifies the horizontal size, in pixels, of the tabs in the tab control.

```
property TabWidth;
```

**Description**

This is inherited property.

See TCustomTabControl.TabWidth for details

**1.23.1.2.1.49 TPaletteTab.Visible Property**

Determines whether the component appears onscreen.

```
property Visible;
```

**Description**

This is inherited property.

See TControl.Visible for details

# 1.24 edcDsnEvents Namespace

## 1.24.1 Classes

The following table lists classes in this documentation.

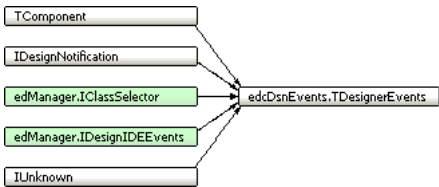
### Classes

Class	Description
TDesignerEvents (🔗 see page 389)	TDesignerEvents provides access to events of active designer and design environment

### 1.24.1.1 TDesignerEvents Class

TDesignerEvents provides access to events of active designer and design environment

#### Class Hierarchy



```
TDesignerEvents = class(TComponent, IDesignNotification, IClassSelector, IDesignIDEEvents, IUnknown);
```

#### File

edcDsnEvents

#### Description

TDesignerEvents implements set of useful IDE interfaces such as

- IDesignNotification
- IClassSelector (🔗 see page 511)
- IDesignIDEEvents (🔗 see page 512)

to provide corresponding behavior on IDE events.

Using this events one can take specific actions on many aspects of design mode.




#### Members

##### IClassSelector Methods

IClassSelector Methods	Description
🔗 ClsChanged (🔗 see page 511)	Called when selected in component palette component class was changed.
🔗 ClsPalChanged (🔗 see page 512)	Called when component palette was changed.















##### IDesignIDEEvents Interface

IDesignIDEEvents Interface	Description
🔗 ActiveDsnChanged (🔗 see page 512)	Called when active designer was changed.
🔗 BeforeRegisterComponent (🔗 see page 512)	Called before registering component class.
🔗 GetGlobalComponents (🔗 see page 512)	Called to get global components.



 GetWorkspaceOrigin ( <a href="#">see page 513</a> )	Called to get workspace origin.
 KeyDown ( <a href="#">see page 513</a> )	Called on key down in active designer.
 KeyPress ( <a href="#">see page 513</a> )	Called on key press in active designer.

## TDesignerEvents Events

### TDesignerEvents Class















TDesignerEvents Class	Description
 OnActiveDsnChanged ( <a href="#">see page 391</a> )	Occurs when DsnManager.ActiveDesigner property is changed
 OnClassChanged ( <a href="#">see page 391</a> )	Occurs when the current selected component class in the component palette is changed.
 OnDesignerClosed ( <a href="#">see page 391</a> )	Occurs when designer is deactivated
 OnDesignerInitialized ( <a href="#">see page 391</a> )	Occurs when designer is initializing.
 OnDsnKeyDown ( <a href="#">see page 392</a> )	Occurs when user presses any key in active designer.
 OnGetGlobalComponents ( <a href="#">see page 392</a> )	Occurs when designer requests global components.
 OnGetWorkspaceOrigin ( <a href="#">see page 392</a> )	Occurs when designer requests workspace origin.
 OnItemDeleted ( <a href="#">see page 392</a> )	Occurs when object is deleted in the active designer
 OnItemInserted ( <a href="#">see page 393</a> )	Occurs when object is inserted in the active designer
 OnItemsModified ( <a href="#">see page 393</a> )	Occurs when object is modified in the active designer
 OnKeyPress ( <a href="#">see page 393</a> )	Occurs when user presses any key in active designer.
 OnPaletteChanged ( <a href="#">see page 393</a> )	Occurs when component palette changed, i.e. registered components is being changed.
 OnRegisterComponent ( <a href="#">see page 394</a> )	Occurs before component will be registered.
 OnSelectionChanged ( <a href="#">see page 394</a> )	Occurs when current selection in the active designer is changed.

### Legend



	Method
	Event

## TDesignerEvents Events





### TDesignerEvents Class



TDesignerEvents Class	Description
 OnActiveDsnChanged ( <a href="#">see page 391</a> )	Occurs when DsnManager.ActiveDesigner property is changed
 OnClassChanged ( <a href="#">see page 391</a> )	Occurs when the current selected component class in the component palette is changed.
 OnDesignerClosed ( <a href="#">see page 391</a> )	Occurs when designer is deactivated
 OnDesignerInitialized ( <a href="#">see page 391</a> )	Occurs when designer is initializing.
 OnDsnKeyDown ( <a href="#">see page 392</a> )	Occurs when user presses any key in active designer.
 OnGetGlobalComponents ( <a href="#">see page 392</a> )	Occurs when designer requests global components.
 OnGetWorkspaceOrigin ( <a href="#">see page 392</a> )	Occurs when designer requests workspace origin.
 OnItemDeleted ( <a href="#">see page 392</a> )	Occurs when object is deleted in the active designer
 OnItemInserted ( <a href="#">see page 393</a> )	Occurs when object is inserted in the active designer
 OnItemsModified ( <a href="#">see page 393</a> )	Occurs when object is modified in the active designer
 OnKeyPress ( <a href="#">see page 393</a> )	Occurs when user presses any key in active designer.
 OnPaletteChanged ( <a href="#">see page 393</a> )	Occurs when component palette changed, i.e. registered components is being changed.
 OnRegisterComponent ( <a href="#">see page 394</a> )	Occurs before component will be registered.
 OnSelectionChanged ( <a href="#">see page 394</a> )	Occurs when current selection in the active designer is changed.

## IClassSelector Methods

IClassSelector Methods	Description
 ClsChanged ( <a href="#">see page 511</a> )	Called when selected in component palette component class was changed.
 ClsPalChanged ( <a href="#">see page 512</a> )	Called when component palette was changed.

## IDesignIDEEvents Interface

IDesignIDEEvents Interface	Description
 ActiveDsnChanged ( <a href="#">see page 512</a> )	Called when active designer was changed.
 BeforeRegisterComponent ( <a href="#">see page 512</a> )	Called before registering component class.
 GetGlobalComponents ( <a href="#">see page 512</a> )	Called to get global components.
 GetWorkspaceOrigin ( <a href="#">see page 513</a> )	Called to get workspace origin.

 KeyDown (see page 513)	Called on key down in active designer.
 KeyPress (see page 513)	Called on key press in active designer.

## 1.24.1.1.1 TDesignerEvents Events

### 1.24.1.1.1.1 TDesignerEvents.OnActiveDsnChanged Event

Occurs when DsnManager.ActiveDesigner property is changed

**property** OnActiveDsnChanged: TNotifyEvent;

#### Description

Write an OnActiveDsnChanged event handler to catch when the active designer is changed.

Sender is the TDesignerEvents (see page 389) himself.

### 1.24.1.1.1.2 TDesignerEvents.OnClassChanged Event

Occurs when the current selected component class in the component palette is changed.

**property** OnClassChanged: TNotifyEvent;

#### Description

Write an OnClassChanged event handler to take some specific action when the current selected component class in the component palette is changed.

Sender is the TDesignerEvents (see page 389) himself.

### 1.24.1.1.1.3 TDesignerEvents.OnDesignerClosed Event

Occurs when designer is deactivated

**property** OnDesignerClosed: TDesignerEvent;

#### Description

Write an OnDesignerClosed event handler to catch when the active designer is about to closed.

OnActiveDsnChanged (see page 391) fires before this event.

Sender is the TDesignerEvents (see page 389) himself.

ADesigner is the corresponding designer.

### 1.24.1.1.1.4 TDesignerEvents.OnDesignerInitialized Event

Occurs when designer is initializing.

**property** OnDesignerInitialized: TDesignerEvent;

#### Description

Write an OnDesignerInitialized event handler to catch when the designer is about to initialized.

Sender is the TDesignerEvents (see page 389) himself.

ADesigner is the corresponding designer.

#### 1.24.1.1.1.5 TDesignerEvents.OnDsnKeyDown Event

Occurs when user presses any key in active designer.

**property** OnDsnKeyDown: TDsnKeyDownEvent;

##### Description

Write an OnDsnKeyDown event handler to take specific action when user presses any key in active designer.

Sender is the corresponding designer.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

#### 1.24.1.1.1.6 TDesignerEvents.OnGetGlobalComponents Event

Occurs when designer requests global components.

**property** OnGetGlobalComponents: TGetGlobalsEvent;

##### Description

Write an OnGetGlobalComponents event handler to fill list of components.

Root is the Root-component is being designed.

List is the list for filling with components

#### 1.24.1.1.1.7 TDesignerEvents.OnGetWorkspaceOrigin Event

Occurs when designer requests workspace origin.

**property** OnGetWorkspaceOrigin: TOnGetPoint;

##### Description

Write an OnGetWorkspaceOrigin event handler to provides information for DesignWindows positioning.

Sender is the TDesignerEvents (see page 389) himself.

P is the topleft point.

#### 1.24.1.1.1.8 TDesignerEvents.OnItemDeleted Event

Occurs when object is deleted in the active designer

**property** OnItemDeleted: TDsnItemEvent;

##### Description

Write an OnItemDeleted event handler to take specific action when item is being deleted.

Sender is the TDesignerEvents (see page 389) himself.

Altem is the item being deleted

#### 1.24.1.1.1.9 TDesignerEvents.OnItemInserted Event

Occurs when object is inserted in the active designer

**property** OnItemInserted: TDsnItemEvent;

##### Description

Write an OnItemInserted event handler to take specific action when item is being inserted.

Sender is the TDesignerEvents (see page 389) himself.

Altem is the item being inserted

#### 1.24.1.1.1.10 TDesignerEvents.OnItemsModified Event

Occurs when object is modified in the active designer

**property** OnItemsModified: TDesignerEvent;

##### Description

Write an OnItemsModified event handler to take specific action when item is being modified.

Sender is the TDesignerEvents (see page 389) himself.

Altem is the item being modified

#### 1.24.1.1.1.11 TDesignerEvents.OnKeyPress Event

Occurs when user presses any key in active designer.

**property** OnKeyPress: TDsnKeyPressEvent;

##### Description

Write an OnKeyPress event handler to make something happen as a result of a single character key press..

Sender is the corresponding designer.

The Key parameter in the OnKeyPress event handler is of type Char.

This is similar to standard TWinControl.OnKeyPress event. See it for details.

#### 1.24.1.1.1.12 TDesignerEvents.OnPaletteChanged Event

Occurs when component palette changed, i.e. registered components is being changed.

**property** OnPaletteChanged: TNotifyEvent;

**Description**

Write an OnPaletteChanged event handler to take some specific action when the component palette is changed.

Sender is the TDesignerEvents (see page 389) himself.

**1.24.1.1.1.13 TDesignerEvents.OnRegisterComponent Event**

Occurs before component will be registered.

**property** OnRegisterComponent: TRegisterComponentEvent;

**Description**

Using this event you may filter components during loading packages.

The AClass parameter is the component class is being registered currently.

The Page parameter is the name of palette page component is being installed to. Change this property to customize output.

The Accept parameter determines whether a component is allowed to be registered.

**1.24.1.1.1.14 TDesignerEvents.OnSelectionChanged Event**

Occurs when current selection in the active designer is changed.

**property** OnSelectionChanged: TNotifyEvent;

**Description**

Write an OnSelectionChanged event handler to take some specific action when current selection in the active designer is changed.

Sender is the TDesignerEvents (see page 389) himself.

## 1.24.2 Types

The following table lists types in this documentation.

**Types**

Type	Description
TDesignerEvent (see page 394)	Type of event handler for designer related events.
TDsnItemEvent (see page 395)	Type of event handler for designed item related events.
TDsnKeyDownEvent (see page 395)	See TDesignerEvents.OnDsnKeyDown Event (see page 392)
TDsnKeyPressEvent (see page 395)	See TDesignerEvents.OnKeyPress Event (see page 393)
TGetGlobalsEvent (see page 395)	See TDesignerEvents.OnGetGlobalComponents Event (see page 392)
TOnGetPoint (see page 395)	See TDesignerEvents.OnGetWorkspaceOrigin Event (see page 392)
TRegisterComponentEvent (see page 396)	See TDesignerEvents.OnRegisterComponent Event (see page 394)

### 1.24.2.1 edcDsnEvents.TDesignerEvent Type

TDesignerEvent = **procedure** (Sender: TObject; ADesigner: IUnknown) **of object**;

**File**

edcDsnEvents

**Description**

Type of event handler for designer related events.

## 1.24.2.2 edcDsnEvents.TDsnItemEvent Type

```
TDsnItemEvent = procedure (Sender, AItem: TObject) of object;
```

**File**

edcDsnEvents

**Description**

Type of event handler for designed item related events.

## 1.24.2.3 edcDsnEvents.TDsnKeyDownEvent Type

```
TDsnKeyDownEvent = procedure (Sender: IDesigner; var Key: Word; Shift: TShiftState) of object;
```

**File**

edcDsnEvents

**Description**

See TDesignerEvents.OnDsnKeyDown Event ([see page 392](#))

## 1.24.2.4 edcDsnEvents.TDsnKeyPressEvent Type

```
TDsnKeyPressEvent = procedure (Sender: IDesigner; var Key: Char) of object;
```

**File**

edcDsnEvents

**Description**

See TDesignerEvents.OnKeyPress Event ([see page 393](#))

## 1.24.2.5 edcDsnEvents.TGetGlobalsEvent Type

```
TGetGlobalsEvent = function (Root: TComponent; const List: TList): Boolean of object;
```

**File**

edcDsnEvents

**Description**

See TDesignerEvents.OnGetGlobalComponents Event ([see page 392](#))

## 1.24.2.6 edcDsnEvents.TOnGetPoint Type

```
TOnGetPoint = procedure (Sender: TObject; var P: TPoint) of object;
```

**File**

edcDsnEvents



**Description**

See TDesignerEvents.OnGetWorkspaceOrigin Event (see page 392)

1.24.2.7 edcDsnEvents.TRegisterComponentEvent Type

```
TRegisterComponentEvent = procedure (ComponentClass: TComponentClass; var Page: string; var
Accept: Boolean) of object;
```

**File**

edcDsnEvents

**Description**

See TDesignerEvents.OnRegisterComponent Event (see page 394)

1.25 edcPropCtrl Namespace

1.25.1 Classes

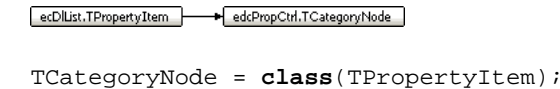
The following table lists classes in this documentation.

Class	Description
TCategoryNode (see page 396)	Determines "Category node" in the TPropertyNodes (see page 442) collection
TCustomInspectorList (see page 399)	TCustomInspectorList is the base class for TInspectorList (see page 413).
TInspectorList (see page 413)	TInspectorList is the descendant of TCustomInspectorList (see page 399) directly using in the Object Inspector
TPropertyNode (see page 439)	Represents single node associated with property
TPropertyNodes (see page 442)	TPropertyNodes represents collection of property
TzDesignerSelections (see page 445)	TzDesignerSelections is the same to TDesignerSelectionList class.

1.25.1.1 TCategoryNode Class

Determines "Category node" in the TPropertyNodes (see page 442) collection

Class Hierarchy



**File**

























edcPropCtrl

**Description**










TCategoryNode used to collect properties by categories in the Object Inspector

## Members










### TPropertyItem Methods

TPropertyItem Methods	Description
 Add (see page 62)	Adds new property item.
  Changed (see page 62)	Called when property item was changed.
  Clear (see page 62)	Deletes all items from the node.
 Create (see page 62)	Creates and initializes a TPropertyItem instance.
 Delete (see page 63)	Deletes Item at index
  Destroy (see page 63)	Destroys an instance of TPropertyItem.
  Expandable (see page 63)	Specifies whether property item can be expanded.
  GetName (see page 63)	Returns name of the item. May be overridden in derived class.
  HasValue (see page 63)	Specifies whether property item has value. For example, category item does not have value.
 IndexOf (see page 63)	Returns index of child item. If Item is not a child returns -1.
 Insert (see page 63)	Adds a property item to the Items (see page 64) array at the position specified by Index.
  IsEqual (see page 63)	Returns True if property items are equal.
  IsRoot (see page 63)	Returns True if the item is root (see page 64) item.
 Move (see page 63)	Changes the position of an item in the Items (see page 64) array.
  Root (see page 64)	Specifies Root item.






### TCategoryNode Class

TCategoryNode Class	Description
  Clear (see page 398)	Does nothing in the TCategoryNode
 Create (see page 398)	Creates and initializes a TCategoryNode instance.
  Expandable (see page 398)	Returns True because TCategoryNode is an expandable node
  GetName (see page 399)	Returns Name of the underlying category passed when TCategoryNode object was created
  HasValue (see page 399)	Returns False, because category item occupies both name and value cells.












### TPropertyItem Properties

TPropertyItem Properties	Description
 Count (see page 64)	Determines count of child items.
 DisplayName (see page 64)	Specifies name displayed on screen
 Expanded (see page 64)	Specifies if node is expanded or not.
 Items (see page 64)	Provides indexed access to the child items.
 Level (see page 64)	Indicates the level of indentation of a item within the property list control..
 Name (see page 64)	Specifies the name of the property node.
 Parent (see page 65)	Indicates the parent property of the node.
 PathName (see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
 Visible (see page 65)	Specifies whether item is selected.

### Legend

	Method
	protected
	virtual
	Property
	read only

### TPropertyItem Methods

TPropertyItem Methods	Description
 Add (see page 62)	Adds new property item.
  Changed (see page 62)	Called when property item was changed.
  Clear (see page 62)	Deletes all items from the node.
 Create (see page 62)	Creates and initializes a TPropertyItem instance.
 Delete (see page 63)	Deletes Item at index
  Destroy (see page 63)	Destroys an instance of TPropertyItem.
  Expandable (see page 63)	Specifies whether property item can be expanded.

<b>GetName</b> ( see page 63)	Returns name of the item. May be overridden in derived class.
<b>HasValue</b> ( see page 63)	Specifies whether property item has value. For example, category item does not have value.
<b>IndexOf</b> ( see page 63)	Returns index of child item. If Item is not a child returns -1.
<b>Insert</b> ( see page 63)	Adds a property item to the Items ( see page 64) array at the position specified by Index.
<b>IsEqual</b> ( see page 63)	Returns True if property items are equal.
<b>IsRoot</b> ( see page 63)	Returns True if the item is root ( see page 64) item.
<b>Move</b> ( see page 63)	Changes the position of an item in the Items ( see page 64) array.
<b>Root</b> ( see page 64)	Specifies Root item.

### TCategoryNode Class

TCategoryNode Class	Description
<b>Clear</b> ( see page 398)	Does nothing in the TCategoryNode
<b>Create</b> ( see page 398)	Creates and initializes a TCategoryNode instance.
<b>Expandable</b> ( see page 398)	Returns True because TCategoryNode is an expandable node
<b>GetName</b> ( see page 399)	Returns Name of the underlying category passed when TCategoryNode object was created
<b>HasValue</b> ( see page 399)	Returns False, because category item occupies both name and value cells.

### TPropertyItem Properties

TPropertyItem Properties	Description
<b>Count</b> ( see page 64)	Determines count of child items.
<b>DisplayName</b> ( see page 64)	Specifies name displayed on screen
<b>Expanded</b> ( see page 64)	Specifies if node is expanded or not.
<b>Items</b> ( see page 64)	Provides indexed access to the child items.
<b>Level</b> ( see page 64)	Indicates the level of indentation of a item within the property list control..
<b>Name</b> ( see page 64)	Specifies the name of the property node.
<b>Parent</b> ( see page 65)	Indicates the parent property of the node.
<b>PathName</b> ( see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
<b>Visible</b> ( see page 65)	Specifies whether item is selected.

## 1.25.1.1.1 TCategoryNode Methods

### 1.25.1.1.1.1 TCategoryNode.Clear Method

Does nothing in the TCategoryNode ( see page 396)

```
procedure Clear; override;
```

#### Description

### 1.25.1.1.1.2 TCategoryNode.Create Constructor

Creates and initializes a TCategoryNode instance.

```
constructor Create(Cat: TPropertyCategory; const ADispName: WideString);
```

#### Description

Use Create to programmatically instantiate a TCategoryNode object.

### 1.25.1.1.1.3 TCategoryNode.Expandable Method

Returns True because TCategoryNode ( see page 396) is an expandable node

```
function Expandable: Boolean; override;
```

#### Description

#### 1.25.1.1.1.4 TCategoryNode.GetName Method

Returns Name of the underlying category passed when TCategoryNode (see page 396) object was created

```
function GetName: string; override;
```

##### Description

#### 1.25.1.1.1.5 TCategoryNode.HasValue Method

Returns False, because category item occupies both name and value cells.

```
function HasValue: Boolean; override;
```

### 1.25.1.2 TCustomInspectorList Class

TCustomInspectorList is the base class for TInspectorList (see page 413).

##### Class Hierarchy



```
TCustomInspectorList = class(TCustomPropList, IDesignNotification);
```

##### File

edcPropCtrl

##### Description

TCustomInspectorList represents dual list control with properties of the selected object(s) in the active designer. Any of these properties are accessible for editing.











It declares and implements all the properties and methods for these purposes.

##### Members



















##### TDualList Methods

TDualList Methods	Description
≡ Create (see page 54)	Creates and initializes an instance of TDualList (see page 52).
≡ CreateEditor (see page 54)	Creates in-place editor. Must be overridden in derived classes.
≡ CreateHandle (see page 54)	Creates underlying screen object.
≡ Destroy (see page 54)	Destroys an instance of TDualList (see page 52).
≡ DoMouseWheel (see page 54)	Processes mouse wheel motion.
≡ DrawCell (see page 55)	Draws a specified cell.
≡ DrawStr (see page 55)	Draws string in the specified rectangle
≡ DrawStrW (see page 55)	Draws Unicode string in the specified rectangle
≡ ExecuteAction (see page 55)	Invokes an action with the component as its target.
≡ FocusEditor (see page 56)	Moves focus from list control to child in-place editor.
≡ IsHeaderItem (see page 56)	Determines if specified item is header.
≡ ItemRect (see page 56)	Returns the rectangle that surrounds the item specified in the Item parameter.
≡ KeyDown (see page 56)	Respond to key press events.
≡ MouseDown (see page 56)	Generates an OnMouseDown event.
≡ MouseMove (see page 56)	Respond to mouse moving over control area..
≡ MouseToItem (see page 57)	Returns item index depending on coordinates specified in Y parameter.
≡ MouseUp (see page 57)	Generates an OnMouseDown event.
≡ Paint (see page 57)	Renders the image of a dual list.
≡ SetItemIndex (see page 57)	Set method of ItemIndex (see page 59) property.
≡ UpdateAction (see page 57)	Updates an action component to reflect the current state of the component.
≡ UpdateEditor (see page 58)	Updates current Editor (see page 59).















**TCustomPropList Class**

<b>TCustomPropList Class</b>	<b>Description</b>
 <b>Create</b> ( <a href="#">see page 49</a> )	Creates and initializes an instance of TDualList.
 <b>CreateItems</b> ( <a href="#">see page 49</a> )	Creates root item.
 <b>Current</b> ( <a href="#">see page 49</a> )	Returns current selected item. If there is no item selected Current returns nil.
 <b>Destroy</b> ( <a href="#">see page 49</a> )	Destroys an instance of TDualList.
 <b>DoPrepareCanvas</b> ( <a href="#">see page 50</a> )	Prepares Canvas ( <a href="#">see page 58</a> ) before painting cell.
 <b>DrawCell</b> ( <a href="#">see page 50</a> )	Draws dual list cell.
 <b>DrawPropCell</b> ( <a href="#">see page 50</a> )	Draws cell content.
 <b>GutterWidth</b> ( <a href="#">see page 50</a> )	Returns gutter width for specified row in list.
 <b>IsHeaderItem</b> ( <a href="#">see page 50</a> )	Determines if specified item is header.
 <b>MouseDown</b> ( <a href="#">see page 50</a> )	Generates an OnMouseUp event.

**TCustomInspectorList Class**









<b>TCustomInspectorList Class</b>	<b>Description</b>
 <b>AcceptProperty</b> ( <a href="#">see page 404</a> )	Called before adding property to the list.
 <b>CopyName</b> ( <a href="#">see page 404</a> )	Copies selected property name to the clipboard.
 <b>CopyValue</b> ( <a href="#">see page 405</a> )	Copies text of the property editor.
 <b>Create</b> ( <a href="#">see page 405</a> )	
 <b>CreateEditor</b> ( <a href="#">see page 405</a> )	Creates TEdit inplace object for the current property
 <b>CreateItems</b> ( <a href="#">see page 405</a> )	Creates root item.
 <b>CutValue</b> ( <a href="#">see page 405</a> )	Cuts text of the property editor.
 <b>DoPrepareCanvas</b> ( <a href="#">see page 405</a> )	Prepares Canvas before painting cell.
 <b>DrawPropCell</b> ( <a href="#">see page 406</a> )	Draws cell content.
 <b>FocusEditor</b> ( <a href="#">see page 406</a> )	Moves focus from list control to child in-place editor.
 <b>GetDesigner</b> ( <a href="#">see page 406</a> )	Indicates if item is header one
 <b>IsPropReadOnly</b> ( <a href="#">see page 406</a> )	Determines whether property editor does not allow changing property value.
 <b>KeyDown</b> ( <a href="#">see page 406</a> )	Responds to key down when TInspectorList ( <a href="#">see page 413</a> ) has focus.
 <b>Loaded</b> ( <a href="#">see page 406</a> )	Updates TCustomInspectorList after loading from stream
 <b>MouseDown</b> ( <a href="#">see page 406</a> )	Updates TCustomInspectorList depending on mouse coordinates
 <b>PasteValue</b> ( <a href="#">see page 406</a> )	Pastes text to the property editor.
 <b>PropValueChanged</b> ( <a href="#">see page 407</a> )	Updates editor and invalidates TCustomInspectorList
 <b>SaveValue</b> ( <a href="#">see page 407</a> )	Saves currently edit value.
 <b>SetItemIndex</b> ( <a href="#">see page 407</a> )	Change active editor
 <b>UpdateEditor</b> ( <a href="#">see page 407</a> )	Updates editor
 <b>UpdateList</b> ( <a href="#">see page 407</a> )	Updates list of properties

**TDualList Properties**






























<b>TDualList Properties</b>	<b>Description</b>
 <b>BorderStyle</b> ( <a href="#">see page 58</a> )	Determines the style of the line drawn around the perimeter of the panel control.
 <b>Canvas</b> ( <a href="#">see page 58</a> )	Provides access to a drawing surface that represents the TDualList ( <a href="#">see page 52</a> ).
 <b>Editor</b> ( <a href="#">see page 59</a> )	In-place editor
 <b>EditorVisible</b> ( <a href="#">see page 59</a> )	Specifies whether editor is visible in the selected row
 <b>ItemCount</b> ( <a href="#">see page 59</a> )	Specifies the number of items in the DualList
 <b>ItemHeight</b> ( <a href="#">see page 59</a> )	Specifies the height, in pixels, of the items in the dual list
 <b>ItemIndex</b> ( <a href="#">see page 59</a> )	Specifies the index of the selected item.
 <b>OnClick</b> ( <a href="#">see page 60</a> )	Occurs when the user clicks the dual list.
 <b>ShowGrid</b> ( <a href="#">see page 60</a> )	Determines whether lines are drawn separating items in the list
 <b>ShowSelFrame</b> ( <a href="#">see page 60</a> )	Specifies whether frame rectangle should be painted around selected item.
 <b>SplitPos</b> ( <a href="#">see page 60</a> )	Specifies width of the first column in pixels (or splitter position)
 <b>TabOrder</b> ( <a href="#">see page 60</a> )	Indicates the position of the control in its parent's tab order.
 <b>TabStop</b> ( <a href="#">see page 60</a> )	Add a summary here...
 <b>TopItem</b> ( <a href="#">see page 60</a> )	Specifies the topmost row that appears in the dual list.

**TCustomPropList Class**

<b>TCustomPropList Class</b>	<b>Description</b>
 <b>cGutter</b> ( <a href="#">see page 50</a> )	Specifies color of gutter background.



 cGutterBnd (see page 51)	Specifies color of border which separates gutter from the rest of control.
 cHighlight (see page 51)	Specifies background color of selected item.
 cHighlightText (see page 51)	Specifies font color of selected item.
 FoldingIcon (see page 51)	Holds folding icon images.
 Items (see page 51)	Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.
 LeftMargin (see page 51)	Specifies left margin.
 LevelWidth (see page 51)	Specifies offset for each level of items.
 ShowGutter (see page 51)	Specifies whether gutter is visible.

## TCustomInspectorList Class





TCustomInspectorList Class	Description
 ByCategories (see page 407)	Specifies arrangement of properties list. If ByCategories is True, all properties are arranged by categories
 Categories (see page 408)	Determines list of the current available categories for the properties of selected object(s). This list does not include empty categories
 cCategories (see page 408)	Specifies font color for category items
 cDefValues (see page 408)	Specifies font color property values which differ from default.
 cEditBackGround (see page 408)	Specifies font color for the in-place editors
 cEditValue (see page 408)	Specifies font color for the text in the in-place editor
 Component (see page 408)	Specifies editable component
 cPropName (see page 409)	Specifies font color for the Property name
 cPropReadOnly (see page 409)	Specifies font color for the read-only property
 cPropReference (see page 409)	Specifies font color for the reference property
 cPropValue (see page 409)	Specifies font color for the property value
 cSubProperty (see page 409)	Specifies font color for the property, that belongs to referenced object
 DefPropNameDraw (see page 409)	Specifies whether custom property name drawing defined in property editors is used.
 Designer (see page 409)	Specifies current active designer
 EditedObject (see page 410)	Specifies editable TPersistent object
 ExpandRefs (see page 410)	Specifies whether reference properties may be expanded
 HiddenCount (see page 410)	Indicates number of properties those are not visible
 HintProps (see page 410)	Provide properties to adjust hints processing.
 IncludeRefs (see page 410)	Specifies whether reference properties should be displayed in the inspector list.
 MarkNonDefault (see page 410)	Specifies if properties those values differ from default will be marked with bold
 PopupListAlign (see page 411)	Specifies alignment of popup list box relative to edit box.
 ReadOnly (see page 411)	Specifies whether properties may be edited.
 SearchPropKey (see page 411)	Property name search key.
 SearchPropMode (see page 411)	Specifies whether inspector list is in property name search mode. This mode is turned on/off when user presses TAB key.
 Selected (see page 411)	Specifies number of selected object in the inspector list.
 SelectedCount (see page 411)	Number of selected objects in active designer. Use property Selected (see page 411)[] to get these objects.
 ShowReadOnly (see page 411)	Specifies whether read-only properties will be displayed in the inspector list
 TypeKinds (see page 411)	Specifies types of properties those will be displayed in the inspector list
 TypeSelector (see page 412)	Specifies kind of inspector list. Property TypeKinds (see page 411) is used only when TypeSelector = tsCustom.





## TCustomPropList Events

## TCustomPropList Class







TCustomPropList Class	Description
 OnDrawPropCell (see page 51)	Draws cell content.
 OnGetCellParams (see page 52)	Occurs to adjust cell properties.

## TCustomInspectorList Class



TCustomInspectorList Class	Description
 OnAcceptCategory (see page 412)	Occurs before adding category to the list.
 OnAcceptProperty (see page 412)	Occurs before adding property to the list.
 OnChangeSelection (see page 412)	Occurs when changing selected object in inspector list.
 OnGetPropReadOnly (see page 413)	Occurs to determine whether property PropEdit is read-only.

 OnPropListUpdated (see page 413)	Occurs after list of properties has been updated
 OnPropValueChanged (see page 413)	Occurs when property value has been changed.
 OnSetPropValueA (see page 413)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 413)	Occurs before changing property value (Unicode version).









**Legend**

	Constructor
	virtual
	protected
	Property
	read only
	Event






















**TCustomPropList Events****TCustomPropList Class**

TCustomPropList Class	Description
 OnDrawPropCell (see page 51)	Draws cell content.
 OnGetCellParams (see page 52)	Occurs to adjust cell properties.



**TCustomInspectorList Class**

TCustomInspectorList Class	Description
 OnAcceptCategory (see page 412)	Occurs before adding category to the list.
 OnAcceptProperty (see page 412)	Occurs before adding property to the list.
 OnChangeSelection (see page 412)	Occurs when changing selected object in inspector list.
 OnGetPropReadOnly (see page 413)	Occurs to determine whether property PropEdit is read-only.
 OnPropListUpdated (see page 413)	Occurs after list of properties has been updated
 OnPropValueChanged (see page 413)	Occurs when property value has been changed.
 OnSetPropValueA (see page 413)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 413)	Occurs before changing property value (Unicode version).














**TDualList Methods**

TDualList Methods	Description
 Create (see page 54)	Creates and initializes an instance of TDualList (see page 52).
 CreateEditor (see page 54)	Creates in-place editor. Must be overridden in derived classes.
 CreateHandle (see page 54)	Creates underlying screen object.
 Destroy (see page 54)	Destroys an instance of TDualList (see page 52).
 DoMouseWheel (see page 54)	Processes mouse wheel motion.
 DrawCell (see page 55)	Draws a specified cell.
 DrawStr (see page 55)	Draws string in the specified rectangle
 DrawStrW (see page 55)	Draws Unicode string in the specified rectangle
 ExecuteAction (see page 55)	Invokes an action with the component as its target.
 FocusEditor (see page 56)	Moves focus from list control to child in-place editor.
 IsHeaderItem (see page 56)	Determines if specified item is header.
 ItemRect (see page 56)	Returns the rectangle that surrounds the item specified in the Item parameter.
 KeyDown (see page 56)	Respond to key press events.
 MouseDown (see page 56)	Generates an OnMouseUp event.
 MouseMove (see page 56)	Respond to mouse moving over control area..
 MouseToItem (see page 57)	Returns item index depending on coordinates specified in Y parameter.
 MouseUp (see page 57)	Generates an OnMouseUp event.
 Paint (see page 57)	Renders the image of a dual list.
 SetItemIndex (see page 57)	Set method of ItemIndex (see page 59) property.
 UpdateAction (see page 57)	Updates an action component to reflect the current state of the component.
 UpdateEditor (see page 58)	Updates current Editor (see page 59).


































**TCustomPropList Class**

TCustomPropList Class	Description
 Create (see page 49)	Creates and initializes an instance of TDualList.
 CreateItems (see page 49)	Creates root item.
 Current (see page 49)	Returns current selected item. If there is no item selected Current returns nil.

















  Destroy (see page 49)	Destroys an instance of TDualList.
  DoPrepareCanvas (see page 50)	Prepares Canvas (see page 58) before painting cell.
  DrawCell (see page 50)	Draws dual list cell.
  DrawPropCell (see page 50)	Draws cell content.
 GutterWidth (see page 50)	Returns gutter width for specified row in list.
  IsHeaderItem (see page 50)	Determines if specified item is header.
  MouseDown (see page 50)	Generates an OnMouseDown event.







### TCustomInspectorList Class

TCustomInspectorList Class	Description
  AcceptProperty (see page 404)	Called before adding property to the list.
 CopyName (see page 404)	Copies selected property name to the clipboard.
 CopyValue (see page 405)	Copies text of the property editor.
  Create (see page 405)	
  CreateEditor (see page 405)	Creates TEditText inplace object for the current property
  CreateItems (see page 405)	Creates root item.
 CutValue (see page 405)	Cuts text of the property editor.
  DoPrepareCanvas (see page 405)	Prepares Canvas before painting cell.
  DrawPropCell (see page 406)	Draws cell content.
  FocusEditor (see page 406)	Moves focus from list control to child in-place editor.
 GetDesigner (see page 406)	Indicates if item is header one
  IsPropReadOnly (see page 406)	Determines whether property editor does not allow changing property value.
  KeyDown (see page 406)	Responds to key down when TInspectorList (see page 413) has focus.
  Loaded (see page 406)	Updates TCustomInspectorList after loading from stream
  MouseDown (see page 406)	Updates TCustomInspectorList depending on mouse coordinates
 PasteValue (see page 406)	Pastes text to the property editor.
 PropValueChanged (see page 407)	Updates editor and invalidates TCustomInspectorList
 SaveValue (see page 407)	Saves currently edit value.
  SetItemIndex (see page 407)	Change active editor
  UpdateEditor (see page 407)	Updates editor
 UpdateList (see page 407)	Updates list of properties




### TDualList Properties

TDualList Properties	Description
 BorderStyle (see page 58)	Determines the style of the line drawn around the perimeter of the panel control.
 Canvas (see page 58)	Provides access to a drawing surface that represents the TDualList (see page 52).
 Editor (see page 59)	In-place editor
 EditorVisible (see page 59)	Specifies whether editor is visible in the selected row
 ItemCount (see page 59)	Specifies the number of items in the DualList
 ItemHeight (see page 59)	Specifies the height, in pixels, of the items in the dual list
 ItemIndex (see page 59)	Specifies the index of the selected item.
 OnClick (see page 60)	Occurs when the user clicks the dual list.
 ShowGrid (see page 60)	Determines whether lines are drawn separating items in the list
 ShowSelFrame (see page 60)	Specifies whether frame rectangle should be painted around selected item.
 SplitPos (see page 60)	Specifies width of the first column in pixels (or splitter position)
 TabOrder (see page 60)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 60)	Add a summary here...
 TopItem (see page 60)	Specifies the topmost row that appears in the dual list.






























### TCustomPropList Class

TCustomPropList Class	Description
 cGutter (see page 50)	Specifies color of gutter background.
 cGutterBnd (see page 51)	Specifies color of border which separates gutter from the rest of control.
 cHighlight (see page 51)	Specifies background color of selected item.
 cHighlightText (see page 51)	Specifies font color of selected item.
 FoldingIcon (see page 51)	Holds folding icon images.
 Items (see page 51)	Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.



 LeftMargin (see page 51)	Specifies left margin.
 LevelWidth (see page 51)	Specifies offset for each level of items.
 ShowGutter (see page 51)	Specifies whether gutter is visible.

## TCustomInspectorList Class

TCustomInspectorList Class	Description
 ByCategories (see page 407)	Specifies arrangement of properties list. If ByCategories is True, all properties are arranged by categories
 Categories (see page 408)	Determines list of the current available categories for the properties of selected object(s). This list does not include empty categories
 cCategories (see page 408)	Specifies font color for category items
 cDefValues (see page 408)	Specifies font color property values which differ from default.
 cEditBackGround (see page 408)	Specifies font color for the in-place editors
 cEditValue (see page 408)	Specifies font color for the text in the in-place editor
 Component (see page 408)	Specifies editable component
 cPropName (see page 409)	Specifies font color for the Property name
 cPropReadOnly (see page 409)	Specifies font color for the read-only property
 cPropReference (see page 409)	Specifies font color for the reference property
 cPropValue (see page 409)	Specifies font color for the property value
 cSubProperty (see page 409)	Specifies font color for the property, that belongs to referenced object
 DefPropNameDraw (see page 409)	Specifies whether custom property name drawing defined in property editors is used.
 Designer (see page 409)	Specifies current active designer
 EditedObject (see page 410)	Specifies editable TPersistent object
 ExpandRefs (see page 410)	Specifies whether reference properties may be expanded
 HiddenCount (see page 410)	Indicates number of properties those are not visible
 HintProps (see page 410)	Provide properties to adjust hints processing.
 IncludeRefs (see page 410)	Specifies whether reference properties should be displayed in the inspector list.
 MarkNonDefault (see page 410)	Specifies if properties those values differ from default will be marked with bold
 PopupListAlign (see page 411)	Specifies alignment of popup list box relative to edit box.
 ReadOnly (see page 411)	Specifies whether properties may be edited.
 SearchPropKey (see page 411)	Property name search key.
 SearchPropMode (see page 411)	Specifies whether inspector list is in property name search mode. This mode is turned on/off when user presses TAB key.
 Selected (see page 411)	Specifies number of selected object in the inspector list.
 SelectedCount (see page 411)	Number of selected objects in active designer. Use property Selected (see page 411)[] to get these objects.
 ShowReadOnly (see page 411)	Specifies whether read-only properties will be displayed in the inspector list
 TypeKinds (see page 411)	Specifies types of properties those will be displayed in the inspector list
 TypeSelector (see page 412)	Specifies kind of inspector list. Property TypeKinds (see page 411) is used only when TypeSelector = tsCustom.

## 1.25.1.2.1 TCustomInspectorList Methods

### 1.25.1.2.1.1 TCustomInspectorList.AcceptProperty Method

Called before adding property to the list.

```
function AcceptProperty(const PropEdit: IProperty; var DisplayName: WideString): Boolean;
dynamic;
```

#### Description

If function result is False property will not be displayed in the inspector list

### 1.25.1.2.1.2 TCustomInspectorList.CopyName Method

Copies selected property name to the clipboard.

```
procedure CopyName;
```

**Description****1.25.1.2.1.3 TCustomInspectorList.CopyValue Method**

Copies text of the property editor.

```
procedure CopyValue;
```

**Description****1.25.1.2.1.4 TCustomInspectorList.Create Constructor**

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

**Description****1.25.1.2.1.5 TCustomInspectorList.CreateEditor Method**

Creates TExtEdit inplace object for the current property

```
function CreateEditor: TCustomEditEx; override;
```

**Description****1.25.1.2.1.6 TCustomInspectorList.CreateItems Method**

Creates root item.

```
function CreateItems: TPropListRoot; override;
```

**Description**

CreateItems creates items storage.

**1.25.1.2.1.7 TCustomInspectorList.CutValue Method**

Cuts text of the property editor.

```
procedure CutValue;
```

**Description****1.25.1.2.1.8 TCustomInspectorList.DoPrepareCanvas Method**

Prepares Canvas before painting cell.

```
procedure DoPrepareCanvas(Node: TPropertyItem; CellType: TCellType; var Alignment: TAlignment); override;
```

**Description**

Node and CellType specifies position in list. Change property of control Canvas and Alignment parameter to change cell painting style.

### 1.25.1.2.1.9 TCustomInspectorList.DrawPropCell Method

Draws cell content.

```
procedure DrawPropCell(const R: TRect; Node: TPropertyItem; CellType: TCellType; Alignment: TAlignment); override;
```

### 1.25.1.2.1.10 TCustomInspectorList.FocusEditor Method

Moves focus from list control to child in-place editor.

```
procedure FocusEditor; override;
```

### 1.25.1.2.1.11 TCustomInspectorList.GetDesigner Method

Indicates if item is header one

```
function GetDesigner: IFormDesigner;
```

#### Description

Header item used as caption for category and has no value.

### 1.25.1.2.1.12 TCustomInspectorList.IsPropReadOnly Method

Determines whether property editor does not allow changing property value.

```
function IsPropReadOnly(const PropEdit: IProperty): Boolean; virtual;
```

### 1.25.1.2.1.13 TCustomInspectorList.KeyDown Method

Responds to key down when TInspectorList (see page 413) has focus.

```
procedure KeyDown(var Key: Word; Shift: TShiftState); override;
```

#### Description

Process two keystrokes:

- VK\_RETURN: Sets (new) value for the current property
- VK\_ESCAPE: ends up editing in the current editor

### 1.25.1.2.1.14 TCustomInspectorList.Loaded Method

Updates TCustomInspectorList (see page 399) after loading from stream

```
procedure Loaded; override;
```

#### Description

### 1.25.1.2.1.15 TCustomInspectorList.MouseDown Method

Updates TCustomInspectorList (see page 399) depending on mouse coordinates

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer); override;
```

#### Description

### 1.25.1.2.1.16 TCustomInspectorList.PasteValue Method

Pastes text to the property editor.

```
procedure PasteValue;
```

**Description****1.25.1.2.1.17 TCustomInspectorList.PropValueChanged Method**

Updates editor and invalidates TCustomInspectorList (see page 399)

```
procedure PropValueChanged;
```

**Description****1.25.1.2.1.18 TCustomInspectorList.SaveValue Method**

Saves currently edit value.

```
procedure SaveValue;
```

**1.25.1.2.1.19 TCustomInspectorList.SetItemIndex Method**

Change active editor

```
procedure SetItemIndex(Value: integer); override;
```

**Description**

If Index parameter is not equal to current

- Sets new value for current property
- Moves focus to the new one

**1.25.1.2.1.20 TCustomInspectorList.UpdateEditor Method**

Updates editor

```
procedure UpdateEditor; override;
```

**Description**

This method is called after changing editor' properties such as color scheme, value and so on.

**1.25.1.2.1.21 TCustomInspectorList.UpdateList Method**

Updates list of properties

```
procedure UpdateList;
```

**Description**

This method updates list of properties after changing on the active designer and so on

**1.25.1.2.2 TCustomInspectorList Properties****1.25.1.2.2.1 TCustomInspectorList.ByCategories Property**

Specifies arrangement of properties list. If ByCategories is True, all properties are arranged by categories

```
property ByCategories: Boolean;
```

**Description**

Set this property to arrange all properties of the selected object(s) by categories.

Default value is False.

#### 1.25.1.2.2.2 TCustomInspectorList.Categories Property

Determines list of the current available categories for the properties of selected object(s).

This list does not include empty categories

**property** Categories: TStrings;

##### Description

#### 1.25.1.2.2.3 TCustomInspectorList.cCategories Property

Specifies font color for category items

**property** cCategories: TColor;

##### Description

Default value is clPurple

#### 1.25.1.2.2.4 TCustomInspectorList.cDefValues Property

Specifies font color property values which differ from default.

**property** cDefValues: TColor;

##### Description

Default value is clNavy.

#### 1.25.1.2.2.5 TCustomInspectorList.cEditBackGround Property

Specifies font color for the in-place editors

**property** cEditBackGround: TColor;

##### Description

Default value is clWindow

#### 1.25.1.2.2.6 TCustomInspectorList.cEditValue Property

Specifies font color for the text in the in-place editor

**property** cEditValue: TColor;

##### Description

Default value is clWindowText

#### 1.25.1.2.2.7 TCustomInspectorList.Component Property

Specifies editable component

**property** Component: TComponent;

##### Description

This property is used for design component without active designer.

Simply set Designer (see page 409) property and Component property and edit properties of the component directly in the Object Inspector.

#### 1.25.1.2.2.8 TCustomInspectorList.cPropName Property

Specifies font color for the Property name

**property** cPropName: TColor;

##### Description

Default value is clBtnText

#### 1.25.1.2.2.9 TCustomInspectorList.cPropReadOnly Property

Specifies font color for the read-only property

**property** cPropReadOnly: TColor;

##### Description

Default value is clBtnText

#### 1.25.1.2.2.10 TCustomInspectorList.cPropReference Property

Specifies font color for the reference property

**property** cPropReference: TColor;

##### Description

Default value is clMaroon

#### 1.25.1.2.2.11 TCustomInspectorList.cPropValue Property

Specifies font color for the property value

**property** cPropValue: TColor;

##### Description

Default value is clNavy

#### 1.25.1.2.2.12 TCustomInspectorList.cSubProperty Property

Specifies font color for the property, that belongs to referenced object

**property** cSubProperty: TColor;

##### Description

Default value is clGreen

#### 1.25.1.2.2.13 TCustomInspectorList.DefPropNameDraw Property

Specifies whether custom property name drawing defined in property editors is used.

**property** DefPropNameDraw: Boolean;

##### Description

It is possible to define property name rendering in any property editor by implementing ICustomPropertyDrawing interface. If you change displayed property names by assigning new values to TPropertyItem.DisplayName (see page 64) this property drawing implementation will paint default property name and TPropertyItem.DisplayName (see page 64) will not be used.

To disable using ICustomPropertyDrawing for rendering property names set DefPropNameDraw to False.

#### 1.25.1.2.2.14 TCustomInspectorList.Designer Property

Specifies current active designer

```
property Designer: TzFormDesigner;
```

#### Description

### 1.25.1.2.2.15 TCustomInspectorList.EditedObject Property

Specifies editable TPersistent object

```
property EditedObject: TPersistent;
```

#### Description

This property is used for design component without active designer.

Simply set Designer (see page 409) property and Component (see page 408) property and edit properties of the component directly in the Object Inspector.

This is the same as Component (see page 408) property only for TPersistent descendants

### 1.25.1.2.2.16 TCustomInspectorList.ExpandRefs Property

Specifies whether reference properties may be expanded

```
property ExpandRefs: Boolean;
```

#### Description

Default value is True

### 1.25.1.2.2.17 TCustomInspectorList.HiddenCount Property

Indicates number of properties those are not visible

```
property HiddenCount: integer;
```

#### Description

Read-only property

### 1.25.1.2.2.18 TCustomInspectorList.HintProps Property

Provide properties to adjust hints processing.

```
property HintProps: TechHintHelper;
```

### 1.25.1.2.2.19 TCustomInspectorList.IncludeRefs Property

Specifies whether reference properties should be displayed in the inspector list.

```
property IncludeRefs: Boolean;
```

#### Description

When IncludeRefs is True, reference properties will be in list even if TypeKinds (see page 411) does not contain tkClass, for example, on the "Events" page in the object inspector

### 1.25.1.2.2.20 TCustomInspectorList.MarkNonDefault Property

Specifies if properties those values differ from default will be marked with bold

```
property MarkNonDefault: Boolean;
```

#### Description

Default value is True.

#### 1.25.1.2.2.21 TCustomInspectorList.PopupListAlign Property

Specifies alignment of popup list box relative to edit box.

```
property PopupListAlign: TAlignment;
```

#### 1.25.1.2.2.22 TCustomInspectorList.ReadOnly Property

Specifies whether properties may be edited.

```
property ReadOnly: Boolean;
```

##### Description

Use this property to enable/disable editing of properties. When ReadOnly is true editor will not be shown for selected item and property can not be edited.

#### 1.25.1.2.2.23 TCustomInspectorList.SearchPropKey Property

Property name search key.

```
property SearchPropKey: string;
```

#### 1.25.1.2.2.24 TCustomInspectorList.SearchPropMode Property

Specifies whether inspector list is in property name search mode. This mode is turned on/off when user presses TAB key.

```
property SearchPropMode: Boolean;
```

#### 1.25.1.2.2.25 TCustomInspectorList.Selected Property

Specifies number of selected object in the inspector list.

```
property Selected [Index: integer]: TPersistent;
```

##### Description

If inspector list is linked to designer this property is equal to number of selected objects in designer.

#### 1.25.1.2.2.26 TCustomInspectorList.SelectedCount Property

Number of selected objects in active designer. Use property Selected ( see page 411)[] to get these objects.

```
property SelectedCount: integer;
```

##### Description

#### 1.25.1.2.2.27 TCustomInspectorList.ShowReadOnly Property

Specifies whether read-only properties will be displayed in the inspector list

```
property ShowReadOnly: Boolean;
```

##### Description

Default value is False

#### 1.25.1.2.2.28 TCustomInspectorList.TypeKinds Property

Specifies types of properties those will be displayed in the inspector list

```
property TypeKinds: TTypeKinds;
```

##### Description

Default value is tkProperties



### 1.25.1.2.2.29 TCustomInspectorList.TypeSelector Property

Specifies kind of inspector list. Property TypeKinds (see page 411) is used only when TypeSelector = tsCustom.

**property** TypeSelector: TTypeSelector;

### 1.25.1.2.3 TCustomInspectorList Events

#### 1.25.1.2.3.1 TCustomInspectorList.OnAcceptCategory Event

Occurs before adding category to the list.

**property** OnAcceptCategory: TAcceptCategoryEvent;

##### Description

Write this event handler to take some specific action before category will be added to the list, for example to localize categories names.

Sender is the TInspectorList (see page 413) object

CategoryName is the name of the category that can be localized

Accept is the flag if category will be added to the list

#### 1.25.1.2.3.2 TCustomInspectorList.OnAcceptProperty Event

Occurs before adding property to the list.

**property** OnAcceptProperty: TAcceptPropertyEvent;

##### Description

Write this event handler to take some specific action before property will be added to the list.

Assign False to Accept parameter to exclude property from the inspector list.

PropEdit is the pointer to the IProperty (see page 446) interface

PropName is the name of property

Accept is the flag if property will be added to the list

#### 1.25.1.2.3.3 TCustomInspectorList.OnChangeSelection Event

Occurs when changing selected object in inspector list.

**property** OnChangeSelection: TChangeSelectionEvent;

##### Description

Write OnChangeSelection event handler to perform any action when selected objects in inspector are to be changed.

It is possible to change selection by creating new IDesignerSelections which will contain another objects.

#### 1.25.1.2.3.4 TCustomInspectorList.OnGetPropReadOnly Event

Occurs to determine whether property PropEdit is read-only.

**property** OnGetPropReadOnly: TGetPropReadOnlyEvent;

#### 1.25.1.2.3.5 TCustomInspectorList.OnPropListUpdated Event

Occurs after list of properties has been updated

**property** OnPropListUpdated: TNotifyEvent;

##### Description

Write this event handler to take some specific action after list of properties has been updated.

#### 1.25.1.2.3.6 TCustomInspectorList.OnPropValueChanged Event

Occurs when property value has been changed.

**property** OnPropValueChanged: TNotifyEvent;

##### Description

Use this event handler to process changing of property value.

#### 1.25.1.2.3.7 TCustomInspectorList.OnSetPropValueA Event

Occurs before changing property value (ANSI version).

**property** OnSetPropValueA: TOnInspSetPropValueEventA;

##### Description

Write OnSetPropValueA to change value to be written to property of selected objects.

#### 1.25.1.2.3.8 TCustomInspectorList.OnSetPropValueW Event

Occurs before changing property value (Unicode version).

**property** OnSetPropValueW: TOnInspSetPropValueEventW;

##### Description

Write OnSetPropValueW to change value to be written to property of selected objects.

### 1.25.1.3 TInspectorList Class

TInspectorList is the descendant of TCustomInspectorList (see page 399) directly using in the Object Inspector

##### Class Hierarchy



```
TInspectorList = class(TCustomInspectorList);
```






















##### File

edcPropCtrl











##### Description

TInspectorList publishes inherited properties to provide wide range of functionality.


















**Members****TDualList Methods**





TDualList Methods	Description
 <b>Create</b> ( <a href="#">see page 54</a> )	Creates and initializes an instance of TDualList ( <a href="#">see page 52</a> ).
 <b>CreateEditor</b> ( <a href="#">see page 54</a> )	Creates in-place editor. Must be overridden in derived classes.
 <b>CreateHandle</b> ( <a href="#">see page 54</a> )	Creates underlying screen object.
 <b>Destroy</b> ( <a href="#">see page 54</a> )	Destroys an instance of TDualList ( <a href="#">see page 52</a> ).
 <b>DoMouseWheel</b> ( <a href="#">see page 54</a> )	Processes mouse wheel motion.
 <b>DrawCell</b> ( <a href="#">see page 55</a> )	Draws a specified cell.
 <b>DrawStr</b> ( <a href="#">see page 55</a> )	Draws string in the specified rectangle
 <b>DrawStrW</b> ( <a href="#">see page 55</a> )	Draws Unicode string in the specified rectangle
 <b>ExecuteAction</b> ( <a href="#">see page 55</a> )	Invokes an action with the component as its target.
 <b>FocusEditor</b> ( <a href="#">see page 56</a> )	Moves focus from list control to child in-place editor.
 <b>IsHeaderItem</b> ( <a href="#">see page 56</a> )	Determines if specified item is header.
 <b>ItemRect</b> ( <a href="#">see page 56</a> )	Returns the rectangle that surrounds the item specified in the Item parameter.
 <b>KeyDown</b> ( <a href="#">see page 56</a> )	Respond to key press events.
 <b>MouseDown</b> ( <a href="#">see page 56</a> )	Generates an OnMouseDown event.
 <b>MouseMove</b> ( <a href="#">see page 56</a> )	Respond to mouse moving over control area..
 <b>MouseToItem</b> ( <a href="#">see page 57</a> )	Returns item index depending on coordinates specified in Y parameter.
 <b>MouseUp</b> ( <a href="#">see page 57</a> )	Generates an OnMouseUp event.
 <b>Paint</b> ( <a href="#">see page 57</a> )	Renders the image of a dual list.
 <b>SetItemIndex</b> ( <a href="#">see page 57</a> )	Set method of ItemIndex ( <a href="#">see page 59</a> ) property.
 <b>UpdateAction</b> ( <a href="#">see page 57</a> )	Updates an action component to reflect the current state of the component.
 <b>UpdateEditor</b> ( <a href="#">see page 58</a> )	Updates current Editor ( <a href="#">see page 59</a> ).

**TCustomPropList Class**















TCustomPropList Class	Description
 <b>Create</b> ( <a href="#">see page 49</a> )	Creates and initializes an instance of TDualList.
 <b>CreateItems</b> ( <a href="#">see page 49</a> )	Creates root item.
 <b>Current</b> ( <a href="#">see page 49</a> )	Returns current selected item. If there is no item selected Current returns nil.
 <b>Destroy</b> ( <a href="#">see page 49</a> )	Destroys an instance of TDualList.
 <b>DoPrepareCanvas</b> ( <a href="#">see page 50</a> )	Prepares Canvas ( <a href="#">see page 58</a> ) before painting cell.
 <b>DrawCell</b> ( <a href="#">see page 50</a> )	Draws dual list cell.
 <b>DrawPropCell</b> ( <a href="#">see page 50</a> )	Draws cell content.
 <b>GutterWidth</b> ( <a href="#">see page 50</a> )	Returns gutter width for specified row in list.
 <b>IsHeaderItem</b> ( <a href="#">see page 50</a> )	Determines if specified item is header.
 <b>MouseDown</b> ( <a href="#">see page 50</a> )	Generates an OnMouseDown event.

**TCustomInspectorList Class**










TCustomInspectorList Class	Description
 <b>AcceptProperty</b> ( <a href="#">see page 404</a> )	Called before adding property to the list.
 <b>CopyName</b> ( <a href="#">see page 404</a> )	Copies selected property name to the clipboard.
 <b>CopyValue</b> ( <a href="#">see page 405</a> )	Copies text of the property editor.
 <b>Create</b> ( <a href="#">see page 405</a> )	
 <b>CreateEditor</b> ( <a href="#">see page 405</a> )	Creates TEdit inplace object for the current property
 <b>CreateItems</b> ( <a href="#">see page 405</a> )	Creates root item.
 <b>CutValue</b> ( <a href="#">see page 405</a> )	Cuts text of the property editor.
 <b>DoPrepareCanvas</b> ( <a href="#">see page 405</a> )	Prepares Canvas before painting cell.
 <b>DrawPropCell</b> ( <a href="#">see page 406</a> )	Draws cell content.
 <b>FocusEditor</b> ( <a href="#">see page 406</a> )	Moves focus from list control to child in-place editor.
 <b>GetDesigner</b> ( <a href="#">see page 406</a> )	Indicates if item is header one
 <b>IsPropReadOnly</b> ( <a href="#">see page 406</a> )	Determines whether property editor does not allow changing property value.
 <b>KeyDown</b> ( <a href="#">see page 406</a> )	Responds to key down when TInspectorList has focus.
 <b>Loaded</b> ( <a href="#">see page 406</a> )	Updates TCustomInspectorList ( <a href="#">see page 399</a> ) after loading from stream
 <b>MouseDown</b> ( <a href="#">see page 406</a> )	Updates TCustomInspectorList ( <a href="#">see page 399</a> ) depending on mouse coordinates
 <b>PasteValue</b> ( <a href="#">see page 406</a> )	Pastes text to the property editor.
 <b>PropValueChanged</b> ( <a href="#">see page 407</a> )	Updates editor and invalidates TCustomInspectorList ( <a href="#">see page 399</a> )

 SaveValue (see page 407)	Saves currently edit value.
 SetItemIndex (see page 407)	Change active editor
 UpdateEditor (see page 407)	Updates editor
 UpdateList (see page 407)	Updates list of properties



















## TDualList Properties












TDualList Properties	Description
 BorderStyle (see page 58)	Determines the style of the line drawn around the perimeter of the panel control.
 Canvas (see page 58)	Provides access to a drawing surface that represents the TDualList (see page 52).
 Editor (see page 59)	In-place editor
 EditorVisible (see page 59)	Specifies whether editor is visible in the selected row
 ItemCount (see page 59)	Specifies the number of items in the DualList
 ItemHeight (see page 59)	Specifies the height, in pixels, of the items in the dual list
 ItemIndex (see page 59)	Specifies the index of the selected item.
 OnClick (see page 60)	Occurs when the user clicks the dual list.
 ShowGrid (see page 60)	Determines whether lines are drawn separating items in the list
 ShowSelFrame (see page 60)	Specifies whether frame rectangle should be painted around selected item.
 SplitPos (see page 60)	Specifies width of the first column in pixels (or splitter position)
 TabOrder (see page 60)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 60)	Add a summary here...
 TopItem (see page 60)	Specifies the topmost row that appears in the dual list.

## TCustomPropList Class





TCustomPropList Class	Description
 cGutter (see page 50)	Specifies color of gutter background.
 cGutterBnd (see page 51)	Specifies color of border which separates gutter from the rest of control.
 cHighlight (see page 51)	Specifies background color of selected item.
 cHighlightText (see page 51)	Specifies font color of selected item.
 FoldingIcon (see page 51)	Holds folding icon images.
 Items (see page 51)	Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.
 LeftMargin (see page 51)	Specifies left margin.
 LevelWidth (see page 51)	Specifies offset for each level of items.
 ShowGutter (see page 51)	Specifies whether gutter is visible.



















































## TCustomInspectorList Class

TCustomInspectorList Class	Description
 ByCategories (see page 407)	Specifies arrangement of properties list. If ByCategories is True, all properties are arranged by categories
 Categories (see page 408)	Determines list of the current available categories for the properties of selected object(s). This list does not include empty categories
 cCategories (see page 408)	Specifies font color for category items
 cDefValues (see page 408)	Specifies font color property values which differ from default.
 cEditBackGround (see page 408)	Specifies font color for the in-place editors
 cEditValue (see page 408)	Specifies font color for the text in the in-place editor
 Component (see page 408)	Specifies editable component
 cPropName (see page 409)	Specifies font color for the Property name
 cPropReadOnly (see page 409)	Specifies font color for the read-only property
 cPropReference (see page 409)	Specifies font color for the reference property
 cPropValue (see page 409)	Specifies font color for the property value
 cSubProperty (see page 409)	Specifies font color for the property, that belongs to referenced object
 DefPropNameDraw (see page 409)	Specifies whether custom property name drawing defined in property editors is used.
 Designer (see page 409)	Specifies current active designer
 EditedObject (see page 410)	Specifies editable TPersistent object
 ExpandRefs (see page 410)	Specifies whether reference properties may be expanded
 HiddenCount (see page 410)	Indicates number of properties those are not visible
 HintProps (see page 410)	Provide properties to adjust hints processing.

 IncludeRefs (see page 410)	Specifies whether reference properties should be displayed in the inspector list.
 MarkNonDefault (see page 410)	Specifies if properties those values differ from default will be marked with bold
 PopupListAlign (see page 411)	Specifies alignment of popup list box relative to edit box.
 ReadOnly (see page 411)	Specifies whether properties may be edited.
 SearchPropKey (see page 411)	Property name search key.
 SearchPropMode (see page 411)	Specifies whether inspector list is in property name search mode. This mode is turned on/off when user presses TAB key.
 Selected (see page 411)	Specifies number of selected object in the inspector list.
 SelectedCount (see page 411)	Number of selected objects in active designer. Use property Selected (see page 411)[] to get these objects.
 ShowReadOnly (see page 411)	Specifies whether read-only properties will be displayed in the inspector list
 TypeKinds (see page 411)	Specifies types of properties those will be displayed in the inspector list
 TypeSelector (see page 412)	Specifies kind of inspector list. Property TypeKinds (see page 411) is used only when TypeSelector = tsCustom.



## TInspectorList Class

<b>TInspectorList Class</b>	<b>Description</b>
 Align (see page 422)	Determines how the control aligns within its container (parent control).
 Anchors (see page 423)	Specifies how the control is anchored to its parent.
 BevelEdges (see page 423)	Specifies which edges of the control are beveled.
 BevelInner (see page 423)	Specifies the cut of the inner bevel.
 BevelKind (see page 424)	Specifies the control's bevel style.
 BevelOuter (see page 424)	Specifies the cut of the outer bevel.
 BiDiMode (see page 424)	Specifies the bi-directional mode for the control.
 BorderStyle (see page 424)	Determines the style of the line drawn around the perimeter of the panel control.
 ByCategories (see page 425)	Specifies arrangement of properties list. If ByCategories is True, all properties are arranged by categories
 cCategories (see page 425)	Specifies font color for category items
 cDefValues (see page 425)	Specifies font color property values which differ from default.
 cEditBackGround (see page 425)	Specifies font color for the in-place editors
 cEditValue (see page 425)	Specifies font color for the text in the in-place editor
 cGutter (see page 425)	Specifies color of gutter background.
 cGutterBnd (see page 425)	Specifies color of border which separates gutter from the rest of control.
 cHighlight (see page 425)	Specifies background color of selected item.
 cHighlightText (see page 426)	Specifies font color of selected item.
 Color (see page 426)	Specifies the background color of the control.
 Component (see page 426)	Specifies editable component
 Constraints (see page 426)	Specifies the size constraints for the control.
 cPropName (see page 426)	Specifies font color for the Property name
 cPropReadOnly (see page 426)	Specifies font color for the read-only property
 cPropReference (see page 427)	Specifies font color for the reference property
 cPropValue (see page 427)	Specifies font color for the property value
 cSubProperty (see page 427)	Specifies font color for the property, that belongs to referenced object
 Ctl3D (see page 427)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DefPropNameDraw (see page 427)	Specifies whether custom property name drawing defined in property editors is used.
 Designer (see page 427)	Specifies current active designer
 DragCursor (see page 428)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 428)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 428)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 EditorVisible (see page 428)	Specifies whether editor is visible in the selected row
 Enabled (see page 428)	Controls whether the control responds to mouse, keyboard, and timer events.
 ExpandRefs (see page 428)	Specifies whether reference properties may be expanded
 FoldingIcon (see page 429)	Holds folding icon images.
 Font (see page 429)	Controls the attributes of text written on or in the control.
 IncludeRefs (see page 429)	Specifies whether reference properties should be displayed in the inspector list.
 ItemHeight (see page 429)	Specifies the height, in pixels, of the items in the dual list









 ItemIndex (see page 429)	Specifies the index of the selected item.
 Items (see page 429)	Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.
 MarkNonDefault (see page 430)	Specifies if properties those values differ from default will be marked with bold
 OnAcceptCategory (see page 430)	Occurs before adding category to the list.
 OnAcceptProperty (see page 430)	Occurs before adding property to the list.
 OnCanResize (see page 430)	Occurs when an attempt is made to resize the control.
 OnChangeSelection (see page 431)	Occurs when changing selected object in inspector list.
 OnClick (see page 431)	Occurs when the user clicks the dual list.
 OnConstrainedResize (see page 431)	Adjust resize constraints.
 OnContextPopup (see page 431)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 432)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 432)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 432)	Occurs when the user drags an object over a control.
 OnDrawPropCell (see page 433)	Draws cell content.
 OnEndDrag (see page 433)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 433)	Occurs when a control receives the input focus.
 OnExit (see page 433)	Occurs when the input focus shifts away from one control to another.
 OnGetCellParams (see page 433)	Occurs to adjust cell properties.
 OnGetPropReadOnly (see page 433)	Occurs to determine whether property PropEdit is read-only.
 OnKeyDown (see page 433)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 434)	Occurs when key pressed.
 OnKeyUp (see page 434)	Occurs when the user releases a key that has been pressed.
 OnMouseDown (see page 434)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 435)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 435)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnPropListUpdated (see page 435)	Occurs after list of properties has been updated
 OnResize (see page 435)	Occurs immediately after the control is resized.
 OnSetPropValueA (see page 435)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 436)	Occurs before changing property value (Unicode version).
 OnStartDrag (see page 436)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 436)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 436)	Determines where a control looks for its color information.
 ParentCtl3D (see page 436)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 436)	Determines where a control looks for its font information.
 ParentShowHint (see page 437)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupListAlign (see page 437)	Specifies alignment of popup list box relative to edit box.
 PopupMenu (see page 437)	Identifies the pop-up menu associated with the control.
 ReadOnly (see page 437)	Specifies whether properties may be edited.
 ShowGrid (see page 437)	Determines whether lines are drawn separating items in the list
 ShowGutter (see page 437)	Specifies whether gutter is visible.
 ShowHint (see page 437)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 ShowReadOnly (see page 437)	Specifies whether read-only properties will be displayed in the inspector list
 ShowSelFrame (see page 438)	Specifies whether frame rectangle should be painted around selected item.
 SplitPos (see page 438)	Specifies width of the first column in pixels (or splitter position)
 TabOrder (see page 438)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 438)	Add a summary here...
 TopItem (see page 438)	Specifies the topmost row that appears in the dual list.
 TypeKinds (see page 438)	Specifies types of properties those will be displayed in the inspector list
 TypeSelector (see page 438)	Specifies kind of inspector list. Property TypeKinds is used only when TypeSelector = tsCustom.
 Visible (see page 439)	Determines whether the component appears on screen.

**TCustomPropList Events**







**TCustomPropList Class**

<b>TCustomPropList Class</b>	<b>Description</b>
 OnDrawPropCell (see page 51)	Draws cell content.
 OnGetCellParams (see page 52)	Occurs to adjust cell properties.



**TCustomInspectorList Class**

<b>TCustomInspectorList Class</b>	<b>Description</b>
 OnAcceptCategory (see page 412)	Occurs before adding category to the list.
 OnAcceptProperty (see page 412)	Occurs before adding property to the list.
 OnChangeSelection (see page 412)	Occurs when changing selected object in inspector list.
 OnGetPropReadOnly (see page 413)	Occurs to determine whether property PropEdit is read-only.
 OnPropListUpdated (see page 413)	Occurs after list of properties has been updated
 OnPropValueChanged (see page 413)	Occurs when property value has been changed.
 OnSetPropValueA (see page 413)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 413)	Occurs before changing property value (Unicode version).









**Legend**

	Constructor
	virtual
	protected
	Property
	read only
	Event
















**TCustomPropList Events****TCustomPropList Class**

<b>TCustomPropList Class</b>	<b>Description</b>
 OnDrawPropCell (see page 51)	Draws cell content.
 OnGetCellParams (see page 52)	Occurs to adjust cell properties.







**TCustomInspectorList Class**

<b>TCustomInspectorList Class</b>	<b>Description</b>
 OnAcceptCategory (see page 412)	Occurs before adding category to the list.
 OnAcceptProperty (see page 412)	Occurs before adding property to the list.
 OnChangeSelection (see page 412)	Occurs when changing selected object in inspector list.
 OnGetPropReadOnly (see page 413)	Occurs to determine whether property PropEdit is read-only.
 OnPropListUpdated (see page 413)	Occurs after list of properties has been updated
 OnPropValueChanged (see page 413)	Occurs when property value has been changed.
 OnSetPropValueA (see page 413)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 413)	Occurs before changing property value (Unicode version).











**TDualList Methods**

<b>TDualList Methods</b>	<b>Description</b>
 Create (see page 54)	Creates and initializes an instance of TDualList (see page 52).
 CreateEditor (see page 54)	Creates in-place editor. Must be overridden in derived classes.
 CreateHandle (see page 54)	Creates underlying screen object.
 Destroy (see page 54)	Destroys an instance of TDualList (see page 52).
 DoMouseWheel (see page 54)	Processes mouse wheel motion.
 DrawCell (see page 55)	Draws a specified cell.
 DrawStr (see page 55)	Draws string in the specified rectangle
 DrawStrW (see page 55)	Draws Unicode string in the specified rectangle
 ExecuteAction (see page 55)	Invokes an action with the component as its target.
 FocusEditor (see page 56)	Moves focus from list control to child in-place editor.
 IsHeaderItem (see page 56)	Determines if specified item is header.
 ItemRect (see page 56)	Returns the rectangle that surrounds the item specified in the Item parameter.
 KeyDown (see page 56)	Respond to key press events.
 MouseDown (see page 56)	Generates an OnMouseDown event.
 MouseMove (see page 56)	Respond to mouse moving over control area..




 MouseToItem (see page 57)	Returns item index depending on coordinates specified in Y parameter.
 MouseUp (see page 57)	Generates an OnMouseUp event.
 Paint (see page 57)	Renders the image of a dual list.
 SetItemIndex (see page 57)	Set method of ItemIndex (see page 59) property.
 UpdateAction (see page 57)	Updates an action component to reflect the current state of the component.
 UpdateEditor (see page 58)	Updates current Editor (see page 59).












**TCustomPropList Class**

TCustomPropList Class	Description
 Create (see page 49)	Creates and initializes an instance of TDualList.
 CreateItems (see page 49)	Creates root item.
 Current (see page 49)	Returns current selected item. If there is no item selected Current returns nil.
 Destroy (see page 49)	Destroys an instance of TDualList.
 DoPrepareCanvas (see page 50)	Prepares Canvas (see page 58) before painting cell.
 DrawCell (see page 50)	Draws dual list cell.
 DrawPropCell (see page 50)	Draws cell content.
 GutterWidth (see page 50)	Returns gutter width for specified row in list.
 IsHeaderItem (see page 50)	Determines if specified item is header.
 MouseDown (see page 50)	Generates an OnMouseUp event.




**TCustomInspectorList Class**

TCustomInspectorList Class	Description
 AcceptProperty (see page 404)	Called before adding property to the list.
 CopyName (see page 404)	Copies selected property name to the clipboard.
 CopyValue (see page 405)	Copies text of the property editor.
 Create (see page 405)	
 CreateEditor (see page 405)	Creates TEditText inplace object for the current property
 CreateItems (see page 405)	Creates root item.
 CutValue (see page 405)	Cuts text of the property editor.
 DoPrepareCanvas (see page 405)	Prepares Canvas before painting cell.
 DrawPropCell (see page 406)	Draws cell content.
 FocusEditor (see page 406)	Moves focus from list control to child in-place editor.
 GetDesigner (see page 406)	Indicates if item is header one
 IsPropReadOnly (see page 406)	Determines whether property editor does not allow changing property value.
 KeyDown (see page 406)	Responds to key down when TInspectorList has focus.
 Loaded (see page 406)	Updates TCustomInspectorList (see page 399) after loading from stream
 MouseDown (see page 406)	Updates TCustomInspectorList (see page 399) depending on mouse coordinates
 PasteValue (see page 406)	Pastes text to the property editor.
 PropValueChanged (see page 407)	Updates editor and invalidates TCustomInspectorList (see page 399)
 SaveValue (see page 407)	Saves currently edit value.
 SetItemIndex (see page 407)	Change active editor
 UpdateEditor (see page 407)	Updates editor
 UpdateList (see page 407)	Updates list of properties










**TDualList Properties**

TDualList Properties	Description
 BorderStyle (see page 58)	Determines the style of the line drawn around the perimeter of the panel control.
 Canvas (see page 58)	Provides access to a drawing surface that represents the TDualList (see page 52).
 Editor (see page 59)	In-place editor
 EditorVisible (see page 59)	Specifies whether editor is visible in the selected row
 ItemCount (see page 59)	Specifies the number of items in the DualList
 ItemHeight (see page 59)	Specifies the height, in pixels, of the items in the dual list
 ItemIndex (see page 59)	Specifies the index of the selected item.
 OnClick (see page 60)	Occurs when the user clicks the dual list.
 ShowGrid (see page 60)	Determines whether lines are drawn separating items in the list
 ShowSelfFrame (see page 60)	Specifies whether frame rectangle should be painted around selected item.
 SplitPos (see page 60)	Specifies width of the first column in pixels (or splitter position)
































 TabOrder (see page 60)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 60)	Add a summary here...
 TopItem (see page 60)	Specifies the topmost row that appears in the dual list.




**TCustomPropList Class**


























<b>TCustomPropList Class</b>	<b>Description</b>
 cGutter (see page 50)	Specifies color of gutter background.
 cGutterBnd (see page 51)	Specifies color of border which separates gutter from the rest of control.
 cHighlight (see page 51)	Specifies background color of selected item.
 cHighlightText (see page 51)	Specifies font color of selected item.
 FoldingIcon (see page 51)	Holds folding icon images.
 Items (see page 51)	Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.
 LeftMargin (see page 51)	Specifies left margin.
 LevelWidth (see page 51)	Specifies offset for each level of items.
 ShowGutter (see page 51)	Specifies whether gutter is visible.

**TCustomInspectorList Class**

<b>TCustomInspectorList Class</b>	<b>Description</b>
 ByCategories (see page 407)	Specifies arrangement of properties list. If ByCategories is True, all properties are arranged by categories
 Categories (see page 408)	Determines list of the current available categories for the properties of selected object(s). This list does not include empty categories
 cCategories (see page 408)	Specifies font color for category items
 cDefValues (see page 408)	Specifies font color property values which differ from default.
 cEditBackGround (see page 408)	Specifies font color for the in-place editors
 cEditValue (see page 408)	Specifies font color for the text in the in-place editor
 Component (see page 408)	Specifies editable component
 cPropName (see page 409)	Specifies font color for the Property name
 cPropReadOnly (see page 409)	Specifies font color for the read-only property
 cPropReference (see page 409)	Specifies font color for the reference property
 cPropValue (see page 409)	Specifies font color for the property value
 cSubProperty (see page 409)	Specifies font color for the property, that belongs to referenced object
 DefPropNameDraw (see page 409)	Specifies whether custom property name drawing defined in property editors is used.
 Designer (see page 409)	Specifies current active designer
 EditedObject (see page 410)	Specifies editable TPersistent object
 ExpandRefs (see page 410)	Specifies whether reference properties may be expanded
 HiddenCount (see page 410)	Indicates number of properties those are not visible
 HintProps (see page 410)	Provide properties to adjust hints processing.
 IncludeRefs (see page 410)	Specifies whether reference properties should be displayed in the inspector list.
 MarkNonDefault (see page 410)	Specifies if properties those values differ from default will be marked with bold
 PopupListAlign (see page 411)	Specifies alignment of popup list box relative to edit box.
 ReadOnly (see page 411)	Specifies whether properties may be edited.
 SearchPropKey (see page 411)	Property name search key.
 SearchPropMode (see page 411)	Specifies whether inspector list is in property name search mode. This mode is turned on/off when user presses TAB key.
 Selected (see page 411)	Specifies number of selected object in the inspector list.
 SelectedCount (see page 411)	Number of selected objects in active designer. Use property Selected (see page 411) to get these objects.
 ShowReadOnly (see page 411)	Specifies whether read-only properties will be displayed in the inspector list
 TypeKinds (see page 411)	Specifies types of properties those will be displayed in the inspector list
 TypeSelector (see page 412)	Specifies kind of inspector list. Property TypeKinds (see page 411) is used only when TypeSelector = tsCustom.

**TInspectorList Class**

<b>TInspectorList Class</b>	<b>Description</b>
 Align (see page 422)	Determines how the control aligns within its container (parent control).
 Anchors (see page 423)	Specifies how the control is anchored to its parent.
 BevelEdges (see page 423)	Specifies which edges of the control are beveled.

 BevellInner (see page 423)	Specifies the cut of the inner bevel.
 BevelKind (see page 424)	Specifies the control's bevel style.
 BevelOuter (see page 424)	Specifies the cut of the outer bevel.
 BiDiMode (see page 424)	Specifies the bi-directional mode for the control.
 BorderStyle (see page 424)	Determines the style of the line drawn around the perimeter of the panel control.
 ByCategories (see page 425)	Specifies arrangement of properties list. If ByCategories is True, all properties are arranged by categories
 cCategories (see page 425)	Specifies font color for category items
 cDefValues (see page 425)	Specifies font color property values which differ from default.
 cEditBackGround (see page 425)	Specifies font color for the in-place editors
 cEditValue (see page 425)	Specifies font color for the text in the in-place editor
 cGutter (see page 425)	Specifies color of gutter background.
 cGutterBnd (see page 425)	Specifies color of border which separates gutter from the rest of control.
 cHighlight (see page 425)	Specifies background color of selected item.
 cHighlightText (see page 426)	Specifies font color of selected item.
 Color (see page 426)	Specifies the background color of the control.
 Component (see page 426)	Specifies editable component
 Constraints (see page 426)	Specifies the size constraints for the control.
 cPropName (see page 426)	Specifies font color for the Property name
 cPropReadOnly (see page 426)	Specifies font color for the read-only property
 cPropReference (see page 427)	Specifies font color for the reference property
 cPropValue (see page 427)	Specifies font color for the property value
 cSubProperty (see page 427)	Specifies font color for the property, that belongs to referenced object
 Ctl3D (see page 427)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 DefPropNameDraw (see page 427)	Specifies whether custom property name drawing defined in property editors is used.
 Designer (see page 427)	Specifies current active designer
 DragCursor (see page 428)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragKind (see page 428)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 428)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 EditorVisible (see page 428)	Specifies whether editor is visible in the selected row
 Enabled (see page 428)	Controls whether the control responds to mouse, keyboard, and timer events.
 ExpandRefs (see page 428)	Specifies whether reference properties may be expanded
 FoldingIcon (see page 429)	Holds folding icon images.
 Font (see page 429)	Controls the attributes of text written on or in the control.
 IncludeRefs (see page 429)	Specifies whether reference properties should be displayed in the inspector list.
 ItemHeight (see page 429)	Specifies the height, in pixels, of the items in the dual list
 ItemIndex (see page 429)	Specifies the index of the selected item.
 Items (see page 429)	Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.
 MarkNonDefault (see page 430)	Specifies if properties those values differ from default will be marked with bold
 OnAcceptCategory (see page 430)	Occurs before adding category to the list.
 OnAcceptProperty (see page 430)	Occurs before adding property to the list.
 OnCanResize (see page 430)	Occurs when an attempt is made to resize the control.
 OnChangeSelection (see page 431)	Occurs when changing selected object in inspector list.
 OnClick (see page 431)	Occurs when the user clicks the dual list.
 OnConstrainedResize (see page 431)	Adjust resize constraints.
 OnContextPopup (see page 431)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 432)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 432)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 432)	Occurs when the user drags an object over a control.
 OnDrawPropCell (see page 433)	Draws cell content.
 OnEndDrag (see page 433)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 433)	Occurs when a control receives the input focus.
 OnExit (see page 433)	Occurs when the input focus shifts away from one control to another.

 OnGetCellParams (see page 433)	Occurs to adjust cell properties.
 OnGetPropReadOnly (see page 433)	Occurs to determine whether property PropEdit is read-only.
 OnKeyDown (see page 433)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 434)	Occurs when key pressed.
 OnKeyUp (see page 434)	Occurs when the user releases a key that has been pressed.
 OnMouseDown (see page 434)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 435)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 435)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnPropListUpdated (see page 435)	Occurs after list of properties has been updated
 OnResize (see page 435)	Occurs immediately after the control is resized.
 OnSetPropValueA (see page 435)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 436)	Occurs before changing property value (Unicode version).
 OnStartDrag (see page 436)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 436)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 436)	Determines where a control looks for its color information.
 ParentCtl3D (see page 436)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 436)	Determines where a control looks for its font information.
 ParentShowHint (see page 437)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupListAlign (see page 437)	Specifies alignment of popup list box relative to edit box.
 PopupMenu (see page 437)	Identifies the pop-up menu associated with the control.
 ReadOnly (see page 437)	Specifies whether properties may be edited.
 ShowGrid (see page 437)	Determines whether lines are drawn separating items in the list
 ShowGutter (see page 437)	Specifies whether gutter is visible.
 ShowHint (see page 437)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 ShowReadOnly (see page 437)	Specifies whether read-only properties will be displayed in the inspector list
 ShowSelFrame (see page 438)	Specifies whether frame rectangle should be painted around selected item.
 SplitPos (see page 438)	Specifies width of the first column in pixels (or splitter position)
 TabOrder (see page 438)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 438)	Add a summary here...
 TopItem (see page 438)	Specifies the topmost row that appears in the dual list.
 TypeKinds (see page 438)	Specifies types of properties those will be displayed in the inspector list
 TypeSelector (see page 438)	Specifies kind of inspector list. Property TypeKinds is used only when TypeSelector = tsCustom.
 Visible (see page 439)	Determines whether the component appears on screen.

### 1.25.1.3.1 TInspectorList Properties

#### 1.25.1.3.1.1 TInspectorList.Align Property

Determines how the control aligns within its container (parent control).

**property** Align;

#### Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to alLeft. The value of alLeft for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is alNone, which means a control remains where it is positioned on a form or panel.

**Tip:** If Align is set to alClient, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

**Note:** To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

### 1.25.1.3.1.2 TInspectorList.Anchors Property

Specifies how the control is anchored to its parent.

**property** Anchors;

#### Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

**Note:** If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

### 1.25.1.3.1.3 TInspectorList.BevelEdges Property

Specifies which edges of the control are beveled.

**property** BevelEdges;

#### Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

### 1.25.1.3.1.4 TInspectorList.BevelInner Property

Specifies the cut of the inner bevel.

**property** BevelInner;

#### Description

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner

bevel appears immediately inside the border.

#### 1.25.1.3.1.5 TInspectorList.BevelKind Property

Specifies the control's bevel style.

```
property BevelKind;
```

##### Description

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

#### 1.25.1.3.1.6 TInspectorList.BevelOuter Property

Specifies the cut of the outer bevel.

```
property BevelOuter;
```

##### Description

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

#### 1.25.1.3.1.7 TInspectorList.BiDiMode Property

Specifies the bi-directional mode for the control.

```
property BiDiMode;
```

##### Description

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

#### 1.25.1.3.1.8 TInspectorList.BorderStyle Property

Determines the style of the line drawn around the perimeter of the panel control.

```
property BorderStyle: TBorderStyle;
```

##### Description

Use BorderStyle to specify whether the panel has a single line drawn around it. These are the possible values:

Value	Meaning
bsNone	No visible border
bsSingle	Single-line border

Do not confuse the line drawn around the panel with the BorderWidth of the panel. The BorderWidth of the panel is the distance between the outer and inner bevels.

### 1.25.1.3.1.9 TInspectorList.ByCategories Property

Specifies arrangement of properties list. If ByCategories is True, all properties are arranged by categories

**property** ByCategories: Boolean;

#### Description

Set this property to arrange all properties of the selected object(s) by categories.

Default value is False.

### 1.25.1.3.1.10 TInspectorList.cCategories Property

Specifies font color for category items

**property** cCategories: TColor;

#### Description

Default value is clPurple

### 1.25.1.3.1.11 TInspectorList.cDefValues Property

Specifies font color property values which differ from default.

**property** cDefValues: TColor;

#### Description

Default value is clNavy.

### 1.25.1.3.1.12 TInspectorList.cEditBackGround Property

Specifies font color for the in-place editors

**property** cEditBackGround: TColor;

#### Description

Default value is clWindow

### 1.25.1.3.1.13 TInspectorList.cEditValue Property

Specifies font color for the text in the in-place editor

**property** cEditValue: TColor;

#### Description

Default value is clWindowText

### 1.25.1.3.1.14 TInspectorList.cGutter Property

Specifies color of gutter background.

**property** cGutter: TColor;

### 1.25.1.3.1.15 TInspectorList.cGutterBnd Property

Specifies color of border which separates gutter from the rest of control.

**property** cGutterBnd: TColor;

### 1.25.1.3.1.16 TInspectorList.cHighlight Property

Specifies background color of selected item.

```
property cHighlight: TColor;
```

#### 1.25.1.3.1.17 TInspectorList.cHighlightText Property

Specifies font color of selected item.

```
property cHighlightText: TColor;
```

#### 1.25.1.3.1.18 TInspectorList.Color Property

Specifies the background color of the control.

```
property Color;
```

##### Description

Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to false.

#### 1.25.1.3.1.19 TInspectorList.Component Property

Specifies editable component

```
property Component: TComponent;
```

##### Description

This property is used for design component without active designer.

Simply set Designer property and Component property and edit properties of the component directly in the Object Inspector.

#### 1.25.1.3.1.20 TInspectorList.Constraints Property

Specifies the size constraints for the control.

```
property Constraints;
```

##### Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

**Warning:** Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

#### 1.25.1.3.1.21 TInspectorList.cPropName Property

Specifies font color for the Property name

```
property cPropName: TColor;
```

##### Description

Default value is clBtnText

#### 1.25.1.3.1.22 TInspectorList.cPropReadOnly Property

Specifies font color for the read-only property

```
property cPropReadOnly: TColor;
```

##### Description

Default value is clBtnText

### 1.25.1.3.1.23 TInspectorList.cPropReference Property

Specifies font color for the reference property

**property** cPropReference: TColor;

#### Description

Default value is clMaroon

### 1.25.1.3.1.24 TInspectorList.cPropValue Property

Specifies font color for the property value

**property** cPropValue: TColor;

#### Description

Default value is clNavy

### 1.25.1.3.1.25 TInspectorList.cSubProperty Property

Specifies font color for the property, that belongs to referenced object

**property** cSubProperty: TColor;

#### Description

Default value is clGreen

### 1.25.1.3.1.26 TInspectorList.Ctl3D Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

**property** Ctl3D;

#### Description

Ctl3D is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later.

On earlier platforms, Ctl3D controlled whether the control had a flat or beveled appearance.

### 1.25.1.3.1.27 TInspectorList.DefPropNameDraw Property

Specifies whether custom property name drawing defined in property editors is used.

**property** DefPropNameDraw: Boolean;

#### Description

It is possible to define property name rendering in any property editor by implementing ICustomPropertyDrawing interface. If you change displayed property names by assigning new values to TPropertyItem.DisplayName (see page 64) this property drawing implementation will paint default property name and TPropertyItem.DisplayName (see page 64) will not be used.

To disable using ICustomPropertyDrawing for rendering property names set DefPropNameDraw to False.

### 1.25.1.3.1.28 TInspectorList.Designer Property

Specifies current active designer

**property** Designer: TzFormDesigner;

#### Description



### 1.25.1.3.1.29 TInspectorList.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

```
property DragCursor;
```

#### Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

**Note:** To make a custom cursor available for the DragCursor property, see the Cursor property.

### 1.25.1.3.1.30 TInspectorList.DragKind Property

Specifies whether the control is being dragged normally or for docking.

```
property DragKind;
```

#### Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

### 1.25.1.3.1.31 TInspectorList.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

```
property DragMode;
```

#### Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

### 1.25.1.3.1.32 TInspectorList.EditorVisible Property

Specifies whether editor is visible in the selected row

```
property EditorVisible: Boolean;
```

#### Description

Specifies whether editor is visible or not in the selected row.

This property may become False when corresponding property node is not a TPropertyNode type. For example when user switches object inspector's "Property" tab in "By Category" mode.

### 1.25.1.3.1.33 TInspectorList.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

```
property Enabled;
```

#### Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

### 1.25.1.3.1.34 TInspectorList.ExpandRefs Property

Specifies whether reference properties may be expanded

```
property ExpandRefs: Boolean;
```

#### Description

Default value is True

### 1.25.1.3.1.35 TInspectorList.FoldingIcon Property

Holds folding icon images.

**property** FoldingIcon: TBitmap;

#### Description

FoldingIcon should contain two images in a row, first - collapse icon (-), second - expand icon (+).

Color of bottom-left pixel is used as mask color.

Folding icon is initialized from resource when control is created at design time.

### 1.25.1.3.1.36 TInspectorList.Font Property

Controls the attributes of text written on or in the control.

**property** Font;

#### Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

### 1.25.1.3.1.37 TInspectorList.IncludeRefs Property

Specifies whether reference properties should be displayed in the inspector list.

**property** IncludeRefs: Boolean;

#### Description

When IncludeRefs is True, reference properties will be in list even if TypeKinds does not contain tkClass, for example, on the "Events" page in the object inspector

### 1.25.1.3.1.38 TInspectorList.ItemHeight Property

Specifies the height, in pixels, of the items in the dual list

**property** ItemHeight: integer;

#### Description

Read ItemHeight to determine the height of the items in the dual list. Set this property to change height of the items.

### 1.25.1.3.1.39 TInspectorList.ItemIndex Property

Specifies the index of the selected item.

**property** ItemIndex: integer;

#### Description

Read ItemIndex to determine which item is selected. The first item in the list has index 0, the second item has index 1, and so on.

Set ItemIndex programmatically to select an item by passing in the index value.

If new value less than 0 and ItemCount > 0 then ItemIndex will be 0,

else if new value exceeds ItemCount-1 then ItemIndex will be ItemCount-1.

Else ItemIndex will be set to new value.

### 1.25.1.3.1.40 TInspectorList.Items Property

Reference to root item. Particular root item class may be different in derived classes. Items are accessible via root item.

```
property Items: TPropListRoot;
```

#### 1.25.1.3.1.41 TInspectorList.MarkNonDefault Property

Specifies if properties those values differ from default will be marked with bold

```
property MarkNonDefault: Boolean;
```

##### Description

Default value is True.

#### 1.25.1.3.1.42 TInspectorList.OnAcceptCategory Property

Occurs before adding category to the list.

```
property OnAcceptCategory: TAcceptCategoryEvent;
```

##### Description

Write this event handler to take some specific action before category will be added to the list, for example to localize categories names.

Sender is the TInspectorList object

CategoryName is the name of the category that can be localized

Accept is the flag if category will be added to the list

#### 1.25.1.3.1.43 TInspectorList.OnAcceptProperty Property

Occurs before adding property to the list.

```
property OnAcceptProperty: TAcceptPropertyEvent;
```

##### Description

Write this event handler to take some specific action before property will be added to the list.

Assign False to Accept parameter to exclude property from the inspector list.

PropEdit is the pointer to the IProperty interface

PropName is the name of property

Accept is the flag if property will be added to the list

#### 1.25.1.3.1.44 TInspectorList.OnCanResize Property

Occurs when an attempt is made to resize the control.

```
property OnCanResize;
```

##### Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

#### 1.25.1.3.1.45 TInspectorList.OnChangeSelection Property

Occurs when changing selected object in inspector list.

```
property OnChangeSelection: TChangeSelectionEvent;
```

##### Description

Write OnChangeSelection event handler to perform any action when selected objects in inspector are to be changed.

It is possible to change selection by creating new IDesignerSelections which will contain another objects.

#### 1.25.1.3.1.46 TInspectorList.OnClick Property

Occurs when the user clicks the dual list.

```
property OnClick;
```

##### Description

OnClick is inherited event from TControl.

Use the OnClick event handler to respond when the user clicks the dual list.

#### 1.25.1.3.1.47 TInspectorList.OnConstrainedResize Property

Adjust resize constraints.

```
property OnConstrainedResize;
```

##### Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

##### Notes

The OnConstrainedResize handler is called immediately after the OnCanResize handler.

#### 1.25.1.3.1.48 TInspectorList.OnContextPopup Property

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

```
property OnContextPopup;
```

##### Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM\_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not

set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

**Note:** Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.

#### 1.25.1.3.1.49 TInspectorList.OnDbClick Property

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

**property** OnDbClick;

##### Description

Use the OnDbClick event to respond to mouse double-clicks.

#### 1.25.1.3.1.50 TInspectorList.OnDragDrop Property

Occurs when the user drops an object being dragged.

**property** OnDragDrop;

##### Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

#### 1.25.1.3.1.51 TInspectorList.OnDragOver Property

Occurs when the user drags an object over a control.

**property** OnDragOver;

##### Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

**Note:** Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event

handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

#### 1.25.1.3.1.52 TInspectorList.OnDrawPropCell Property

Draws cell content.

```
property OnDrawPropCell: TCustomPropDrawEvent;
```

##### Description

Write OnDrawPropCell event handler to implement custom drawing of the cell content. If OnDrawPropCell event is assigned control does not perform default drawing, but it prepares Canvas for drawing.

#### 1.25.1.3.1.53 TInspectorList.OnEndDrag Property

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

```
property OnEndDrag;
```

##### Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

#### 1.25.1.3.1.54 TInspectorList.OnEnter Property

Occurs when a control receives the input focus.

```
property OnEnter;
```

##### Description

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

#### 1.25.1.3.1.55 TInspectorList.OnExit Property

Occurs when the input focus shifts away from one control to another.

```
property OnExit;
```

##### Description

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

#### 1.25.1.3.1.56 TInspectorList.OnGetCellParams Property

Occurs to adjust cell properties.

```
property OnGetCellParams: TGetCellParamsEvent;
```

##### Description

Write OnGetCellParams event handler to change cell properties by assigning new values to Alignment parameter and to Pen, Font and Brush of the property list Canvas.

#### 1.25.1.3.1.57 TInspectorList.OnGetPropReadOnly Property

Occurs to determine whether property PropEdit is read-only.

```
property OnGetPropReadOnly: TGetPropReadOnlyEvent;
```

#### 1.25.1.3.1.58 TInspectorList.OnKeyDown Property

Occurs when a user presses any key while the control has focus.

**property** OnKeyDown;

#### Description

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys, and pressed mouse buttons.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

### 1.25.1.3.1.59 TInspectorList.OnKeyPress Property

Occurs when key pressed.

**property** OnKeyPress;

#### Description

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

### 1.25.1.3.1.60 TInspectorList.OnKeyUp Property

Occurs when the user releases a key that has been pressed.

**property** OnKeyUp;

#### Description

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

### 1.25.1.3.1.61 TInspectorList.OnMouseDown Property

Occurs when the user presses a mouse button with the mouse pointer over a control.

**property** OnMouseDown;

#### Description

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a

mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

### 1.25.1.3.1.62 TInspectorList.OnMouseMove Property

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

**property** OnMouseMove;

#### Description

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

### 1.25.1.3.1.63 TInspectorList.OnMouseUp Property

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

**property** OnMouseUp;

#### Description

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

### 1.25.1.3.1.64 TInspectorList.OnPropListUpdated Property

Occurs after list of properties has been updated

**property** OnPropListUpdated: TNotifyEvent;

#### Description

Write this event handler to take some specific action after list of properties has been updated.

### 1.25.1.3.1.65 TInspectorList.OnResize Property

Occurs immediately after the control is resized.

**property** OnResize;

#### Description

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

### 1.25.1.3.1.66 TInspectorList.OnSetPropValueA Property

Occurs before changing property value (ANSI version).

**property** OnSetPropValueA: TOnInspSetPropValueEventA;

#### Description

Write OnSetPropValueA to change value to be written to property of selected objects.



### 1.25.1.3.1.67 TInspectorList.OnSetPropValueW Property

Occurs before changing property value (Unicode version).

**property** OnSetPropValueW: TOnInspSetPropValueEventW;

#### Description

Write OnSetPropValueW to change value to be written to property of selected objects.

### 1.25.1.3.1.68 TInspectorList.OnStartDrag Property

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

**property** OnStartDrag;

#### Description

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itse

### 1.25.1.3.1.69 TInspectorList.ParentBiDiMode Property

Specifies whether the control uses its parent's BiDiMode.

**property** ParentBiDiMode;

### 1.25.1.3.1.70 TInspectorList.ParentColor Property

Determines where a control looks for its color information.

**property** ParentColor;

### 1.25.1.3.1.71 TInspectorList.ParentCtl3D Property

Determines where a component looks to determine if it should appear three dimensional.

**property** ParentCtl3D;

#### Description

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

### 1.25.1.3.1.72 TInspectorList.ParentFont Property

Determines where a control looks for its font information.

```
property ParentFont;
```

#### 1.25.1.3.1.73 TInspectorList.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

```
property ParentShowHint;
```

#### 1.25.1.3.1.74 TInspectorList.PopupListAlign Property

Specifies alignment of popup list box relative to edit box.

```
property PopupListAlign: TAlignment;
```

#### 1.25.1.3.1.75 TInspectorList.PopupMenu Property

Identifies the pop-up menu associated with the control.

```
property PopupMenu;
```

##### Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

#### 1.25.1.3.1.76 TInspectorList.ReadOnly Property

Specifies whether properties may be edited.

```
property ReadOnly: Boolean;
```

##### Description

Use this property to enable/disable editing of properties. When ReadOnly is true editor will not be shown for selected item and property can not be edited.

#### 1.25.1.3.1.77 TInspectorList.ShowGrid Property

Determines whether lines are drawn separating items in the list

```
property ShowGrid: Boolean;
```

##### Description

Specifies whether horizontal grid lines is visible.

Set ShowGrid to True to add lines that separate the items in the dual list.

#### 1.25.1.3.1.78 TInspectorList.ShowGutter Property

Specifies whether gutter is visible.

```
property ShowGutter: Boolean;
```

#### 1.25.1.3.1.79 TInspectorList.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

#### 1.25.1.3.1.80 TInspectorList.ShowReadOnly Property

Specifies whether read-only properties will be displayed in the inspector list

```
property ShowReadOnly: Boolean;
```

##### Description

Default value is False

### 1.25.1.3.1.81 TInspectorList.ShowSelFrame Property

Specifies whether frame rectangle should be painted around selected item.

```
property ShowSelFrame: Boolean;
```

### 1.25.1.3.1.82 TInspectorList.SplitPos Property

Specifies width of the first column in pixels (or splitter position)

```
property SplitPos: integer;
```

#### Description

Specifies width of the first column (or splitter position). Width of the second column equals to Width - SplitPos.

Set SplitPos programmatically to change width of columns.

Minimal value of SplitPos is 20;

### 1.25.1.3.1.83 TInspectorList.TabOrder Property

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

#### Description

It is inherited property from TWinControl. See the TWinControl.TabOrder for description.

### 1.25.1.3.1.84 TInspectorList.TabStop Property

Add a summary here...

```
property TabStop;
```

#### Description

It is inherited property from TWinControl. See the TWinControl.Stop for description.

### 1.25.1.3.1.85 TInspectorList.TopItem Property

Specifies the topmost row that appears in the dual list.

```
property TopItem: integer;
```

#### Description

When TopItem is changed, the dual list scrolls vertically so that the specified row is topmost in the view.

### 1.25.1.3.1.86 TInspectorList.TypeKinds Property

Specifies types of properties those will be displayed in the inspector list

```
property TypeKinds: TTypeKinds;
```

#### Description

Default value is tkProperties

### 1.25.1.3.1.87 TInspectorList.TypeSelector Property

Specifies kind of inspector list. Property TypeKinds is used only when TypeSelector = tsCustom.

```
property TypeSelector: TTypeSelector;
```

### 1.25.1.3.1.88 TInspectorList.Visible Property

Determines whether the component appears on screen.

```
property Visible;
```

#### Description

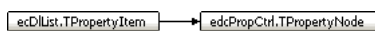
Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

## 1.25.1.4 TPropertyNode Class

Represents single node associated with property

#### Class Hierarchy



```
TPropertyNode = class(TPropertyItem);
```

#### File

edcPropCtrl

#### Description

#### Members

#### TPropertyItem Methods






















TPropertyItem Methods	Description
➤ Add (see page 62)	Adds new property item.
➤ Changed (see page 62)	Called when property item was changed.
➤ Clear (see page 62)	Deletes all items from the node.
➤ Create (see page 62)	Creates and initializes a TPropertyItem instance.
➤ Delete (see page 63)	Deletes Item at index
➤ Destroy (see page 63)	Destroys an instance of TPropertyItem.
➤ Expandable (see page 63)	Specifies whether property item can be expanded.
➤ GetName (see page 63)	Returns name of the item. May be overridden in derived class.
➤ HasValue (see page 63)	Specifies whether property item has value. For example, category item does not have value.
➤ IndexOf (see page 63)	Returns index of child item. If Item is not a child returns -1.
➤ Insert (see page 63)	Adds a property item to the Items (see page 64) array at the position specified by Index.
➤ IsEqual (see page 63)	Returns True if property items are equal.
➤ IsRoot (see page 63)	Returns True if the item is root (see page 64) item.
➤ Move (see page 63)	Changes the position of an item in the Items (see page 64) array.
➤ Root (see page 64)	Specifies Root item.

#### TPropertyNode Class
















TPropertyNode Class	Description
➤ Create (see page 441)	Creates and initializes a TPropertyNode instance.
➤ Expandable (see page 441)	Specifies whether property item can be expanded.
➤ GetName (see page 441)	Returns name of the property.
➤ Owner (see page 441)	Returns the Owner of the TPropertyNode.
➤ ReflectModified (see page 441)	ReflectModified used for refreshing node with reference type property

#### TPropertyItem Properties






TPropertyItem Properties	Description
➤ Count (see page 64)	Determines count of child items.

 DisplayName (  see page 64)	Specifies name displayed on screen
 Expanded (  see page 64)	Specifies if node is expanded or not.
  Items (  see page 64)	Provides indexed access to the child items.
  Level (  see page 64)	Indicates the level of indentation of a item within the property list control..
  Name (  see page 64)	Specifies the name of the property node.
  Parent (  see page 65)	Indicates the parent property of the node.
  PathName (  see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
 Visible (  see page 65)	Specifies whether item is selected.




















































## TPropertyNode Class

TPropertyNode Class	Description
  Editor (  see page 442)	Represents the interface that the Object Inspector uses to communicate with this property editor
  IsDefault (  see page 442)	Indicates if property has default value.
  IsReference (  see page 442)	Indicates whether the property is reference.
  IsSubProperty (  see page 442)	Indicates whether the property is a sub-property.
  PropInfo (  see page 442)	Returns TPropInfo associated with this property
















## Legend

	Method
	protected
	virtual
	Property
	read only











## TPropertyItem Methods











TPropertyItem Methods	Description
 Add (  see page 62)	Adds new property item.
   Changed (  see page 62)	Called when property item was changed.
   Clear (  see page 62)	Deletes all items from the node.
 Create (  see page 62)	Creates and initializes a TPropertyItem instance.
 Delete (  see page 63)	Deletes Item at index
   Destroy (  see page 63)	Destroys an instance of TPropertyItem.
   Expandable (  see page 63)	Specifies whether property item can be expanded.
   GetName (  see page 63)	Returns name of the item. May be overridden in derived class.
   HasValue (  see page 63)	Specifies whether property item has value. For example, category item does not have value.
 IndexOf (  see page 63)	Returns index of child item. If Item is not a child returns -1.
 Insert (  see page 63)	Adds a property item to the Items (  see page 64) array at the position specified by Index.
   IsEqual (  see page 63)	Returns True if property items are equal.
   IsRoot (  see page 63)	Returns True if the item is root (  see page 64) item.
 Move (  see page 63)	Changes the position of an item in the Items (  see page 64) array.
   Root (  see page 64)	Specifies Root item.

## TPropertyNode Class











TPropertyNode Class	Description
 Create (  see page 441)	Creates and initializes a TPropertyNode instance.
   Expandable (  see page 441)	Specifies whether property item can be expanded.
   GetName (  see page 441)	Returns name of the property.
  Owner (  see page 441)	Returns the Owner of the TPropertyNode.
 ReflectModified (  see page 441)	ReflectModified used for refreshing node with reference type property

## TPropertyItem Properties

TPropertyItem Properties	Description
  Count (  see page 64)	Determines count of child items.
 DisplayName (  see page 64)	Specifies name displayed on screen
 Expanded (  see page 64)	Specifies if node is expanded or not.
  Items (  see page 64)	Provides indexed access to the child items.

 <b>Level</b> (  see page 64)	Indicates the level of indentation of a item within the property list control..
 <b>Name</b> (  see page 64)	Specifies the name of the property node.
 <b>Parent</b> (  see page 65)	Indicates the parent property of the node.
 <b>PathName</b> (  see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
 <b>Visible</b> (  see page 65)	Specifies whether item is selected.

### TPropertyNode Class

TPropertyNode Class	Description
 <b>Editor</b> (  see page 442)	Represents the interface that the Object Inspector uses to communicate with this property editor
 <b>IsDefault</b> (  see page 442)	Indicates if property has default value.
 <b>IsReference</b> (  see page 442)	Indicates whether the property is reference.
 <b>IsSubProperty</b> (  see page 442)	Indicates whether the property is a sub-property.
 <b>PropInfo</b> (  see page 442)	Returns PPropInfo associated with this property

## 1.25.1.4.1 TPropertyNode Methods

### 1.25.1.4.1.1 TPropertyNode.Create Constructor

Creates and initializes a TPropertyNode instance.

```
constructor Create(const Prop: IProperty; const ADispName: WideString);
```

#### Description

Use Create to programmatically instantiate a TPropertyNode object.

### 1.25.1.4.1.2 TPropertyNode.Expandable Method

Specifies whether property item can be expanded.

```
function Expandable: Boolean; override;
```

### 1.25.1.4.1.3 TPropertyNode.GetName Method

Returns name of the property.

```
function GetName: string; override;
```

### 1.25.1.4.1.4 TPropertyNode.Owner Method

Returns the Owner of the TPropertyNode ( see page 439).

```
function Owner: TObject;
```


#### Description

### 1.25.1.4.1.5 TPropertyNode.ReflectModified Method

ReflectModified used for refreshing node with reference type property

```
function ReflectModified: Boolean;
```

#### Description

Whenever value of reference property has been changed it must change value of appropriate TPropertyNode ( see page 439).

## 1.25.1.4.2 TPropertyNode Properties

### 1.25.1.4.2.1 TPropertyNode.Editor Property

Represents the interface that the Object Inspector uses to communicate with this property editor

```
property Editor: IProperty;
```

#### Description

The Object Inspector uses this property to interact with certain property editor.

### 1.25.1.4.2.2 TPropertyNode.IsDefault Property

Indicates if property has default value.

```
property IsDefault: Boolean;
```

### 1.25.1.4.2.3 TPropertyNode.IsReference Property

Indicates whether the property is reference.

```
property IsReference: Boolean;
```

### 1.25.1.4.2.4 TPropertyNode.IsSubProperty Property

Indicates whether the property is a sub-property.

```
property IsSubProperty: Boolean;
```

### 1.25.1.4.2.5 TPropertyNode.PropInfo Property

Returns PPropInfo associated with this property

```
property PropInfo: PPropInfo;
```

## 1.25.1.5 TPropertyNodes Class

TPropertyNodes represents collection of property

#### Class Hierarchy



```
TPropertyNodes = class(TPropListRoot);
```

#### File

edcPropCtrl






#### Description

TPropertyNodes represents collection of property associated with the certain TCustomInspectorList (see page 399).











#### Members

##### TPropertyItem Methods


TPropertyItem Methods	Description
➤ Add (see page 62)	Adds new property item.
➤ V Changed (see page 62)	Called when property item was changed.
➤ V Clear (see page 62)	Deletes all items from the node.
➤ Create (see page 62)	Creates and initializes a TPropertyItem instance.
➤ Delete (see page 63)	Deletes Item at index
➤ V Destroy (see page 63)	Destroys an instance of TPropertyItem.
➤ V Expandable (see page 63)	Specifies whether property item can be expanded.
➤ V GetName (see page 63)	Returns name of the item. May be overridden in derived class.
➤ V HasValue (see page 63)	Specifies whether property item has value. For example, category item does not have value.
➤ IndexOf (see page 63)	Returns index of child item. If Item is not a child returns -1.

 Insert (see page 63)	Adds a property item to the Items (see page 64) array at the position specified by Index.
 IsEqual (see page 63)	Returns True if property items are equal.
 IsRoot (see page 63)	Returns True if the item is root (see page 64) item.
 Move (see page 63)	Changes the position of an item in the Items (see page 64) array.
 Root (see page 64)	Specifies Root item.










**TPropListRoot Class**

TPropListRoot Class	Description
 BeginUpdate (see page 67)	Suspends updating of property list.
 Changed (see page 67)	Called when property item was changed.
 Create (see page 67)	Creates and initializes a TPropListRoot instance.
 Destroy (see page 68)	Destroys an instance of TPropListRoot.
 EndUpdate (see page 68)	Re-enables screen repainting.
 ExpandItem (see page 68)	Called when item is to be expanded.
 ExpIndexOf (see page 68)	Returns visible index (expanded) of specified property node.
 RestoreState (see page 68)	Restores previously saved state.
 SaveState (see page 68)	Saves state, i.e. expanded items and select item.
 UpdateList (see page 68)	Updates property list.




**TPropertyNodes Class**

TPropertyNodes Class	Description
 Clear (see page 445)	Deletes all items from the node.
 Create (see page 445)	Creates and initializes a TPropertyNodes instance.
 Destroy (see page 445)	Destroys an instance of TPropertyNodes.
 ExpandItem (see page 445)	Called when item is to be expanded.
 GetProps (see page 445)	Fills list of the property nodes according to Components and Designer passed as parameters






**TPropertyItem Properties**

TPropertyItem Properties	Description
 Count (see page 64)	Determines count of child items.
 DisplayName (see page 64)	Specifies name displayed on screen
 Expanded (see page 64)	Specifies if node is expanded or not.
 Items (see page 64)	Provides indexed access to the child items.
 Level (see page 64)	Indicates the level of indentation of a item within the property list control..
 Name (see page 64)	Specifies the name of the property node.
 Parent (see page 65)	Indicates the parent property of the node.
 PathName (see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
 Visible (see page 65)	Specifies whether item is selected.

**TPropListRoot Class**

TPropListRoot Class	Description
 ExpCount (see page 68)	Returns count of child nodes and their nodes recursively
 ExplItems (see page 68)	Provides indexed access to the list of visible items.
 Owner (see page 69)	Specifies custom property list - owner of property items collection.

**Legend**

	Method
	protected
	virtual
	Property
	read only

**TPropertyItem Methods**

TPropertyItem Methods	Description
 Add (see page 62)	Adds new property item.
 Changed (see page 62)	Called when property item was changed.
 Clear (see page 62)	Deletes all items from the node.



Create (see page 62)	Creates and initializes a TPropertyItem instance.
Delete (see page 63)	Deletes Item at index
Destroy (see page 63)	Destroys an instance of TPropertyItem.
Expandable (see page 63)	Specifies whether property item can be expanded.
GetName (see page 63)	Returns name of the item. May be overridden in derived class.
HasValue (see page 63)	Specifies whether property item has value. For example, category item does not have value.
IndexOf (see page 63)	Returns index of child item. If Item is not a child returns -1.
Insert (see page 63)	Adds a property item to the Items (see page 64) array at the position specified by Index.
IsEqual (see page 63)	Returns True if property items are equal.
IsRoot (see page 63)	Returns True if the item is root (see page 64) item.
Move (see page 63)	Changes the position of an item in the Items (see page 64) array.
Root (see page 64)	Specifies Root item.

**TPropListRoot Class**

TPropListRoot Class	Description
BeginUpdate (see page 67)	Suspends updating of property list.
Changed (see page 67)	Called when property item was changed.
Create (see page 67)	Creates and initializes a TPropListRoot instance.
Destroy (see page 68)	Destroys an instance of TPropListRoot.
EndUpdate (see page 68)	Re-enables screen repainting.
ExpandItem (see page 68)	Called when item is to be expanded.
ExpIndexOf (see page 68)	Returns visible index (expanded) of specified property node.
RestoreState (see page 68)	Restores previously saved state.
SaveState (see page 68)	Saves state, i.e. expanded items and select item.
UpdateList (see page 68)	Updates property list.

**TPropertyNodes Class**

TPropertyNodes Class	Description
Clear (see page 445)	Deletes all items from the node.
Create (see page 445)	Creates and initializes a TPropertyNodes instance.
Destroy (see page 445)	Destroys an instance of TPropertyNodes.
ExpandItem (see page 445)	Called when item is to be expanded.
GetProps (see page 445)	Fills list of the property nodes according to Components and Designer passed as parameters

**TPropertyItem Properties**

TPropertyItem Properties	Description
Count (see page 64)	Determines count of child items.
DisplayName (see page 64)	Specifies name displayed on screen
Expanded (see page 64)	Specifies if node is expanded or not.
Items (see page 64)	Provides indexed access to the child items.
Level (see page 64)	Indicates the level of indentation of a item within the property list control..
Name (see page 64)	Specifies the name of the property node.
Parent (see page 65)	Indicates the parent property of the node.
PathName (see page 65)	Returns path of the item. Path is combined from the item name and all parent names.
Visible (see page 65)	Specifies whether item is selected.

**TPropListRoot Class**

TPropListRoot Class	Description
ExpCount (see page 68)	Returns count of child nodes and their nodes recursively
ExplItems (see page 68)	Provides indexed access to the list of visible items.
Owner (see page 69)	Specifies custom property list - owner of property items collection.

**1.25.1.5.1 TPropertyNodes Methods**

### 1.25.1.5.1.1 TPropertyNodes.Clear Method

Deletes all items from the node.

```
procedure Clear; override;
```

### 1.25.1.5.1.2 TPropertyNodes.Create Constructor

Creates and initializes a TPropertyNodes instance.

```
constructor Create(AOwner: TDualList);
```

**Description**

Use Create to programmatically instantiate a TPropertyNodes object.

### 1.25.1.5.1.3 TPropertyNodes.Destroy Destructor

Destroys an instance of TPropertyNodes.

```
destructor Destroy; override;
```

**Description**

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

### 1.25.1.5.1.4 TPropertyNodes.ExpandItem Method

Called when item is to be expanded.

```
procedure ExpandItem(Item: TPropertyItem); override;
```

### 1.25.1.5.1.5 TPropertyNodes.GetProps Method

Fills list of the property nodes according to Components and Designer passed as parameters

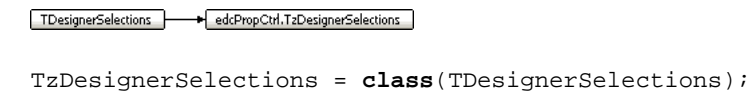
```
procedure GetProps(Components: IDesignerSelections; const Designer: IFormDesigner);
```

**Description**

## 1.25.1.6 TzDesignerSelections Class

TzDesignerSelections is the same to TDesignerSelectionList class.

**Class Hierarchy**



**File**

edcPropCtrl

## 1.25.2 Structs, Records, Enums

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

**Enumerations**

Enumeration	Description
TTypeSelector (🔗 see page 446)	Specifies kind of selector list.

Legend

	Enumeration
-----------------------------------------------------------------------------------	-------------

1.25.2.1 edcPropCtrl.TTypeSelector Enumeration

Specifies kind of selector list.

```
TTypeSelector = (  
    tsProperties,  
    tsEvents,  
    tsCustom  
);
```

File

edcPropCtrl

Members

Members	Description
tsProperties	Only properties (tkProperties)
tsEvents	Only events (tkEvents)
tsCustom	Custom property kinds defined by TCustomInspectorList.TypeKinds Property ( <a href="#">see page 411</a> ).

1.25.3 Types

The following table lists types in this documentation.

Types

Type	Description
IFormDesigner ( <a href="#">see page 446</a> )	IFormDesigner is an alias for IDesigner
IProperty ( <a href="#">see page 446</a> )	IProperty is an alias for TPropertyEditor
TAcceptCategoryEvent ( <a href="#">see page 447</a> )	See TCustomInspectorList.OnAcceptCategory Event ( <a href="#">see page 412</a> )
TAcceptPropertyEvent ( <a href="#">see page 447</a> )	See TCustomInspectorList.OnAcceptProperty Event ( <a href="#">see page 412</a> )
TChangeSelectionEvent ( <a href="#">see page 447</a> )	See TCustomInspectorList.OnChangeSelection Event ( <a href="#">see page 412</a> )
TGetPropReadOnlyEvent ( <a href="#">see page 447</a> )	See TCustomInspectorList.OnGetPropReadOnly Event ( <a href="#">see page 413</a> )
TOnInspSetPropValueEventA ( <a href="#">see page 447</a> )	See TCustomInspectorList.OnSetPropValueA Event ( <a href="#">see page 413</a> )
TOnInspSetPropValueEventW ( <a href="#">see page 448</a> )	See TCustomInspectorList.OnSetPropValueW Event ( <a href="#">see page 413</a> )

1.25.3.1 edcPropCtrl.IFormDesigner Type

IFormDesigner is an alias for IDesigner

```
IFormDesigner = IDesigner;
```

File

edcPropCtrl

Description

1.25.3.2 edcPropCtrl.IProperty Type

IProperty is an alias for TPropertyEditor

```
IProperty = TPropertyEditor;
```

**File**

edcPropCtrl

**Description**

### 1.25.3.3 edcPropCtrl.TAcceptCategoryEvent Type

```
TAcceptCategoryEvent = procedure (Sender: TObject; var CategoryName: WideString; var
Accept: boolean) of object;
```

**File**

edcPropCtrl

**Description**

See TCustomInspectorList.OnAcceptCategory Event (see page 412)

### 1.25.3.4 edcPropCtrl.TAcceptPropertyEvent Type

```
TAcceptPropertyEvent = procedure (const PropEdit: IProperty; var PropName: WideString; var
Accept: Boolean) of object;
```

**File**

edcPropCtrl

**Description**

See TCustomInspectorList.OnAcceptProperty Event (see page 412)

### 1.25.3.5 edcPropCtrl.TChangeSelectionEvent Type

See TCustomInspectorList.OnChangeSelection Event (see page 412)

```
TChangeSelectionEvent = procedure (Sender: TObject; var Selection: IDesignerSelections) of
object;
```

**File**

edcPropCtrl

### 1.25.3.6 edcPropCtrl.TGetPropReadOnlyEvent Type

See TCustomInspectorList.OnGetPropReadOnly Event (see page 413)

```
TGetPropReadOnlyEvent = procedure (Sender: TObject; const PropEdit: IProperty; var
IsReadOnly: Boolean) of object;
```

**File**

edcPropCtrl

### 1.25.3.7 edcPropCtrl.TOnInspSetPropValueEventA Type

See TCustomInspectorList.OnSetPropValueA Event (see page 413)

```
TOnInspSetPropValueEventA = procedure (Sender: TObject; Node: TPropertyNode; var Value:
string) of object;
```

File

edcPropCtrl

1.25.3.8 edcPropCtrl.TOnInspSetPropValueEventW Type

See TCustomInspectorList.OnSetPropValueW Event (see page 413)

TOnInspSetPropValueEventW = **procedure** (Sender: TObject; Node: TPropertyNode; **var** Value: WideString) **of object**;

File

edcPropCtrl

1.26 edcPropEdit Namespace

1.26.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TCustomPropertyEdit (see page 448)	Base class for controls to edit specified property
TPropertyEdit (see page 457)	Edit control to edit specified property.
TPropertyNameProperty (see page 463)	TPropertyNameProperty is the editor for string property with name "PropertyName"

1.26.1.1 TCustomPropertyEdit Class

Base class for controls to edit specified property

Class Hierarchy



TCustomPropertyEdit = **class**(TCustomEditEx);

File

edcPropEdit

Description

TCustomPropertyEdit is used as the base class for TPropertyEdit (see page 457).







































It introduces all the functionality for editing specified property.

Members

























TUnicodeEdit Methods

TUnicodeEdit Methods	Description
🔧 Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
🔧 Destroy (see page 108)	Destroys an instance of TUnicodeEdit.



































**TBtnEdit Class**

TBtnEdit Class	Description
  AdjustClientRect (see page 73)	Overrides the inherited method.
  ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
  Create (see page 73)	Creates and initializes a TBtnEdit (see page 70) instance.
  CreateParams (see page 73)	Overrides the inherited method.
  CreateWnd (see page 73)	Overrides the inherited method.
  Destroy (see page 74)	Destroys an instance of TBtnEdit (see page 70)
  EndTracking (see page 74)	Called after finishing the mouse tracking
  KeyDown (see page 74)	Overrides the inherited method.
  KeyPress (see page 74)	Overrides the inherited method.
  MouseMove (see page 74)	Overrides the inherited method.
  MouseUp (see page 75)	Overrides the inherited method.
  Paint (see page 75)	Overrides the base rendering method.
  PaintBtnGlyph (see page 75)	Renders the image of the button.
  PaintStatus (see page 75)	Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.
  PaintWindow (see page 75)	Overrides the inherited method.
  PtInButton (see page 75)	Checks if specified point is over the button
  StartTracking (see page 76)	Calls immediately after user pushes the button.
  StopTracking (see page 76)	Calls immediately after user releases the button.
  TrackButton (see page 76)	Controls tracking button process.





**TCustomEditEx Class**

TCustomEditEx Class	Description
  AcceptListValue (see page 82)	Active pop-up window.
  ButtonClick (see page 82)	Simulates a button click, as if the user had clicked the button.
 CloseUp (see page 82)	Generates an OnCloseUp (see page 85) event and makes some other actions.
  Create (see page 82)	Creates and initializes a TBtnEdit instance.
  Destroy (see page 82)	Destroys an instance of TBtnEdit
  DoDropDownKeys (see page 83)	Works as a method dispatcher depending on parameter values.
 DropDown (see page 83)	Generates an OnDropDown (see page 85) event.
  EndTracking (see page 83)	Called after finishing the mouse tracking
  KeyPress (see page 83)	Respond to keyboard input.
  MouseDown (see page 83)	Overrides inherited method
  MouseMove (see page 83)	Overrides the inherited method.
  PaintBtnGlyph (see page 84)	Overrides inherited method
  StartTracking (see page 84)	Overrides inherited property











**TCustomPropertyEdit Class**

TCustomPropertyEdit Class	Description
  AcceptListValue (see page 453)	Called when item was selected in drop-down list.
  ButtonClick (see page 453)	Called when button was clicked.
  Change (see page 453)	Reflects changes in property editor to property
  ChangePropertyValue (see page 453)	Writes current text to property.
  Create (see page 453)	Creates and initializes a TCustomPropertyEdit instance.
  DblClick (see page 453)	Calls corresponding property editor
  DoEdit (see page 454)	Calls corresponding property editor
  DoExit (see page 454)	Sets value to corresponding property
  DropDown (see page 454)	Opens drop down list.
  GetEditor (see page 454)	Gets the corresponding editor for the property
  GetStr (see page 454)	Adds string to the pick list of the TCustomPropertyEdit
  GetWStr (see page 454)	Adds Unicode string to the pick list of the TCustomPropertyEdit
  MouseDown (see page 454)	Overrides inherited method
  Notification (see page 455)	Responds to notifications that components are being created or destroyed.
  PaintStatus (see page 455)	Paints status area. This method is called only if StatusWidth is greater 0.
  SetValue (see page 455)	Sets Value to the property editor and invokes OnChange if assigned.
  UpdateEditState (see page 455)	Updates all the properties of the TCustomPropertyEdit object






**TUnicodeEdit Properties**

<b>TUnicodeEdit Properties</b>	<b>Description</b>
 IsUnicode (see page 108)	Specifies whether control is Unicode edit.
 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).








**TBtnEdit Class**

<b>TBtnEdit Class</b>	<b>Description</b>
 Alignment (see page 76)	Determines how the text is aligned within the editor control.
 ButtonVisible (see page 76)	Specifies if button-like rectangle at the right edge of the control is visible.
 ButtonWidth (see page 77)	Specifies width in pixels of the button.
  Canvas (see page 77)	Provides access to the drawing surface of the TBtnEdit (see page 70).
 MultiLine (see page 77)	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth (see page 77)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
 WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

**TCustomEditEx Class**

<b>TCustomEditEx Class</b>	<b>Description</b>
 ActiveList (see page 84)	Specifies current popup control.
 EditStyle (see page 84)	Specifies edit style of the button
 ListAlign (see page 84)	Specifies relative align of popup control.
  PickList (see page 84)	Determines the drop-down list.





**TCustomPropertyEdit Class**

<b>TCustomPropertyEdit Class</b>	<b>Description</b>
 AcceptTab (see page 455)	Specifies whether edit control should process TAB key.
 Component (see page 456)	Specifies owner of the property will be edited
 Designer (see page 456)	Specifies active designer.
 PropertyEditor (see page 456)	Specifies interface for property editor (IProperty)
 PropertyName (see page 456)	Specifies name of the property which will be edited
 ReadOnly (see page 456)	Specifies whether property can not be edited.
 TypeKinds (see page 456)	Set of types of editable properties.

**TBtnEdit Events****TBtnEdit Class**

<b>TBtnEdit Class</b>	<b>Description</b>
 OnButtonClick (see page 78)	Occurs when the user clicks the button.




**TCustomEditEx Class**




<b>TCustomEditEx Class</b>	<b>Description</b>
 OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
 OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
 OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
 OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.

**TCustomPropertyEdit Class**

<b>TCustomPropertyEdit Class</b>	<b>Description</b>
 OnSetPropValueA (see page 457)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 457)	Occurs before changing property value (Unicode version).

**Legend**





	Constructor
	virtual
	protected

	Property
	read only
	Event

**TBtEdit Events****TBtEdit Class**

TBtEdit Class	Description
 OnButtonClick (see page 78)	Occurs when the user clicks the button.

**TCustomEditEx Class**

TCustomEditEx Class	Description
 OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
 OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
 OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
 OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.




















**TCustomPropertyEdit Class**

TCustomPropertyEdit Class	Description
 OnSetPropValueA (see page 457)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 457)	Occurs before changing property value (Unicode version).








**TUnicodeEdit Methods**

TUnicodeEdit Methods	Description
 Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
 Destroy (see page 108)	Destroys an instance of TUnicodeEdit.













**TBtEdit Class**

TBtEdit Class	Description
 AdjustClientRect (see page 73)	Overrides the inherited method.
 ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
 Create (see page 73)	Creates and initializes a TBtEdit (see page 70) instance.
 CreateParams (see page 73)	Overrides the inherited method.
 CreateWnd (see page 73)	Overrides the inherited method.
 Destroy (see page 74)	Destroys an instance of TBtEdit (see page 70)
 EndTracking (see page 74)	Called after finishing the mouse tracking
 KeyDown (see page 74)	Overrides the inherited method.
 KeyPress (see page 74)	Overrides the inherited method.
 MouseMove (see page 74)	Overrides the inherited method.
 MouseUp (see page 75)	Overrides the inherited method.
 Paint (see page 75)	Overrides the base rendering method.
 PaintBtnGlyph (see page 75)	Renders the image of the button.
 PaintStatus (see page 75)	Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.
 PaintWindow (see page 75)	Overrides the inherited method.
 PtInButton (see page 75)	Checks if specified point is over the button
 StartTracking (see page 76)	Calls immediately after user pushes the button.
 StopTracking (see page 76)	Calls immediately after user releases the button.
 TrackButton (see page 76)	Controls tracking button process.



































**TCustomEditEx Class**

TCustomEditEx Class	Description
 AcceptListValue (see page 82)	Active pop-up window.
 ButtonClick (see page 82)	Simulates a button click, as if the user had clicked the button.
 CloseUp (see page 82)	Generates an OnCloseUp (see page 85) event and makes some other actions.
 Create (see page 82)	Creates and initializes a TBtEdit instance.
 Destroy (see page 82)	Destroys an instance of TBtEdit
 DoDropDownKeys (see page 83)	Works as a method dispatcher depending on parameter values.
 DropDown (see page 83)	Generates an OnDropDown (see page 85) event.











  EndTracking (see page 83)	Called after finishing the mouse tracking
  KeyPress (see page 83)	Respond to keyboard input.
  MouseDown (see page 83)	Overrides inherited method
  MouseMove (see page 83)	Overrides the inherited method.
  PaintBtnGlyph (see page 84)	Overrides inherited method
  StartTracking (see page 84)	Overrides inherited property



















**TCustomPropertyEdit Class**

TCustomPropertyEdit Class	Description
  AcceptListValue (see page 453)	Called when item was selected in drop-down list.
  ButtonClick (see page 453)	Called when button was clicked.
  Change (see page 453)	Reflects changes in property editor to property
  ChangePropertyValue (see page 453)	Writes current text to property.
  Create (see page 453)	Creates and initializes a TCustomPropertyEdit instance.
  DblClick (see page 453)	Calls corresponding property editor
  DoEdit (see page 454)	Calls corresponding property editor
  DoExit (see page 454)	Sets value to corresponding property
  DropDown (see page 454)	Opens drop down list.
  GetEditor (see page 454)	Gets the corresponding editor for the property
  GetStr (see page 454)	Adds string to the pick list of the TCustomPropertyEdit
  GetWStr (see page 454)	Adds Unicode string to the pick list of the TCustomPropertyEdit
  MouseDown (see page 454)	Overrides inherited method
  Notification (see page 455)	Responds to notifications that components are being created or destroyed.
  PaintStatus (see page 455)	Paints status area. This method is called only if StatusWidth is greater 0.
  SetValue (see page 455)	Sets Value to the property editor and invokes OnChange if assigned.
  UpdateEditState (see page 455)	Updates all the properties of the TCustomPropertyEdit object









**TUnicodeEdit Properties**

TUnicodeEdit Properties	Description
  IsUnicode (see page 108)	Specifies whether control is Unicode edit.
  SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
  Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
  TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).







**TBtnEdit Class**





TBtnEdit Class	Description
  Alignment (see page 76)	Determines how the text is aligned within the editor control.
  ButtonVisible (see page 76)	Specifies if button-like rectangle at the right edge of the control is visible.
  ButtonWidth (see page 77)	Specifies width in pixels of the button.
  Canvas (see page 77)	Provides access to the drawing surface of the TBtnEdit (see page 70).
  MultiLine (see page 77)	Designates a multiline edit control. The default is single-line edit control.
  StatusWidth (see page 77)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
  WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
  WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
  WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.

**TCustomEditEx Class**

TCustomEditEx Class	Description
  ActiveList (see page 84)	Specifies current popup control.
  EditStyle (see page 84)	Specifies edit style of the button
  ListAlign (see page 84)	Specifies relative align of popup control.
  PickList (see page 84)	Determines the drop-down list.

**TCustomPropertyEdit Class**

TCustomPropertyEdit Class	Description
  AcceptTab (see page 455)	Specifies whether edit control should process TAB key.
  Component (see page 456)	Specifies owner of the property will be edited
  Designer (see page 456)	Specifies active designer.

 PropertyEditor (see page 456)	Specifies interface for property editor (IProperty)
 PropertyName (see page 456)	Specifies name of the property which will be edited
 ReadOnly (see page 456)	Specifies whether property can not be edited.
 TypeKinds (see page 456)	Set of types of editable properties.

## 1.26.1.1.1 TCustomPropertyEdit Methods

### 1.26.1.1.1.1 TCustomPropertyEdit.AcceptListValue Method

Called when item was selected in drop-down list.

```
procedure AcceptListValue(var ListValue: string); override;
```

#### Description

### 1.26.1.1.1.2 TCustomPropertyEdit.ButtonClick Method

Called when button was clicked.

```
procedure ButtonClick; override;
```

#### Description

### 1.26.1.1.1.3 TCustomPropertyEdit.Change Method

Reflects changes in property editor to property

```
procedure Change; override;
```

#### Description

If property attributes include paAutoUpdate then sets editor's value to property and invokes OnChange event

### 1.26.1.1.1.4 TCustomPropertyEdit.ChangePropertyValue Method

Writes current text to property.

```
procedure ChangePropertyValue; virtual;
```

#### Description

Override ChangePropertyValue to perform any actions before updating property value.

### 1.26.1.1.1.5 TCustomPropertyEdit.Create Constructor

Creates and initializes a TCustomPropertyEdit instance.

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

#### Description

Use Create to programmatically instantiate a TCustomPropertyEdit component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

### 1.26.1.1.1.6 TCustomPropertyEdit.DbClick Method

Calls corresponding property editor

```
procedure DbClick; override;
```

**Description****1.26.1.1.1.7 TCustomPropertyEdit.DoEdit Method**

Calls corresponding property editor

```
procedure DoEdit; dynamic;
```

**Description**

This method is called after double clicking on the editor or after mouse click on the editor's button

**1.26.1.1.1.8 TCustomPropertyEdit.DoExit Method**

Sets value to corresponding property

```
procedure DoExit; override;
```

**Description****1.26.1.1.1.9 TCustomPropertyEdit.DropDown Method**

Opens drop down list.

```
procedure DropDown; override;
```

**Description****1.26.1.1.1.10 TCustomPropertyEdit.GetEditor Method**

Gets the corresponding editor for the property

```
procedure GetEditor;
```

**Description**

By means of this method there is accessible a manual updating after loading new packages.

**1.26.1.1.1.11 TCustomPropertyEdit.GetStr Method**

Adds string to the pick list of the TCustomPropertyEdit ([↗](#) see page 448)

```
procedure GetStr(const S: string);
```

**Description****1.26.1.1.1.12 TCustomPropertyEdit.GetWStr Method**

Adds Unicode string to the pick list of the TCustomPropertyEdit ([↗](#) see page 448)

```
procedure GetWStr(const S: WideString);
```

**Description****1.26.1.1.1.13 TCustomPropertyEdit.MouseDown Method**

Overrides inherited method

```
procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X: Integer; Y: Integer);  
override;
```

**Description**

Closes drop-down list up and calls inherited method.

**1.26.1.1.1.14 TCustomPropertyEdit.Notification Method**

Responds to notifications that components are being created or destroyed.

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

**Description**

If Operation is opRemove

- and AComponent is Designer (see page 456) then zeroing Designer (see page 456) property
- else if AComponent is Component (see page 456) then zeroing Component (see page 456) property and calls GetEditor (see page 454) after that

**1.26.1.1.1.15 TCustomPropertyEdit.PaintStatus Method**

Paints status area. This method is called only if StatusWidth is greater 0.

```
procedure PaintStatus(Canvas: TCanvas; Rect: TRect); override;
```

**1.26.1.1.1.16 SetValue Method****1.26.1.1.1.16.1 TCustomPropertyEdit.SetValue Method (WideString)**

Sets Value to the property editor and invokes OnChange if assigned.

```
procedure SetValue(Value: WideString); overload;
```

**Description**

Use this method to set Unicode property value and update edit control.

**1.26.1.1.1.16.2 TCustomPropertyEdit.SetValue Method (string)**

Sets Value to the property editor and invokes OnChange if assigned.

```
procedure SetValue(Value: string); overload;
```

**Description**

Use this method to set property value and update edit control.

**1.26.1.1.1.17 TCustomPropertyEdit.UpdateEditState Method**

Updates all the properties of the TCustomPropertyEdit (see page 448) object

```
procedure UpdateEditState(Reset: Boolean);
```

**Description**

Updates all the properties of the TCustomPropertyEdit (see page 448) object such as EditStyle, Text, Enabled, PropertyEditor (see page 456) itself and so on.

**1.26.1.1.2 TCustomPropertyEdit Properties****1.26.1.1.2.1 TCustomPropertyEdit.AcceptTab Property**

Specifies whether edit control should process TAB key.

```
property AcceptTab: Boolean;
```

#### 1.26.1.1.2.2 TCustomPropertyEdit.Component Property

Specifies owner of the property will be edited

```
property Component: TComponent;
```

##### Description

#### 1.26.1.1.2.3 TCustomPropertyEdit.Designer Property

Specifies active designer.

```
property Designer: TzFormDesigner;
```

##### Description

This property is used for some property editors demanding transfer IDesigner as parameter

#### 1.26.1.1.2.4 TCustomPropertyEdit.PropertyEditor Property

Specifies interface for property editor (IProperty)

```
property PropertyEditor: IProperty;
```

##### Description

The Object Inspector uses the methods on the IProperty interface to interact with property editors.

#### 1.26.1.1.2.5 TCustomPropertyEdit.PropertyName Property

Specifies name of the property which will be edited

```
property PropertyName: string;
```

##### Description

#### 1.26.1.1.2.6 TCustomPropertyEdit.ReadOnly Property

Specifies whether property can not be edited.

```
property ReadOnly: Boolean;
```

##### Description

#### 1.26.1.1.2.7 TCustomPropertyEdit.TypeKinds Property

Set of types of editable properties.

```
property TypeKinds: TTypeKinds;
```

##### Description

This property used to determine if assigned property editor does will be accessible or not.

If assigned property editor does not suit this TypeKinds, it will be inaccessible.

### 1.26.1.1.3 TCustomPropertyEdit Events

1.26.1.1.3.1 TCustomPropertyEdit.OnSetPropValueA Event

Occurs before changing property value (ANSI version).

**property** OnSetPropValueA: TOnSetPropValueEventA;

**Description**

Write OnSetPropValueA to change (see page 453) value to be written to property.

1.26.1.1.3.2 TCustomPropertyEdit.OnSetPropValueW Event

Occurs before changing property value (Unicode version).

**property** OnSetPropValueW: TOnSetPropValueEventW;

**Description**

Write OnSetPropValueW to change (see page 453) value to be written to property.

1.26.1.2 TPropertyEdit Class

Edit control to edit specified property.

Class Hierarchy



TPropertyEdit = **class**(TCustomPropertyEdit);

**File**

edcPropEdit

**Description**

TPropertyEdit is the immediate descendant of the TCustomPropertyEdit (see page 448) type.

It publishes some of the TCustomPropertyEdit (see page 448) protected property.

Edit control to edit specified property. This control uses property editors (IProperty).

Style of the control depends on property editor and may be: simple, ellipsis or combo-box.

Style and some other settings are initialized from property editor.

Use Component (see page 462) and PropertyName (see page 463) properties to specify which property to edit.

To register new property editor use standard Delphi function RegisterPropertyEditor.

Members

TUnicodeEdit Methods

TUnicodeEdit Methods	Description
Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
Destroy (see page 108)	Destroys an instance of TUnicodeEdit.

TBtnEdit Class

TBtnEdit Class	Description
AdjustClientRect (see page 73)	Overrides the inherited method.
ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
Create (see page 73)	Creates and initializes a TBtnEdit (see page 70) instance.

<b>CreateParams</b> ( <a href="#">see page 73</a> )	Overrides the inherited method.
<b>CreateWnd</b> ( <a href="#">see page 73</a> )	Overrides the inherited method.
<b>Destroy</b> ( <a href="#">see page 74</a> )	Destroys an instance of TBtnEdit ( <a href="#">see page 70</a> )
<b>EndTracking</b> ( <a href="#">see page 74</a> )	Called after finishing the mouse tracking
<b>KeyDown</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
<b>KeyPress</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
<b>MouseMove</b> ( <a href="#">see page 74</a> )	Overrides the inherited method.
<b>MouseUp</b> ( <a href="#">see page 75</a> )	Overrides the inherited method.
<b>Paint</b> ( <a href="#">see page 75</a> )	Overrides the base rendering method.
<b>PaintBtnGlyph</b> ( <a href="#">see page 75</a> )	Renders the image of the button.
<b>PaintStatus</b> ( <a href="#">see page 75</a> )	Paints status area. This method is called only if StatusWidth ( <a href="#">see page 77</a> ) is greater 0.
<b>PaintWindow</b> ( <a href="#">see page 75</a> )	Overrides the inherited method.
<b>PtInButton</b> ( <a href="#">see page 75</a> )	Checks if specified point is over the button
<b>StartTracking</b> ( <a href="#">see page 76</a> )	Calls immediately after user pushes the button.
<b>StopTracking</b> ( <a href="#">see page 76</a> )	Calls immediately after user releases the button.
<b>TrackButton</b> ( <a href="#">see page 76</a> )	Controls tracking button process.

**TCustomEditEx Class**




<b>TCustomEditEx Class</b>	<b>Description</b>
<b>AcceptListValue</b> ( <a href="#">see page 82</a> )	Active pop-up window.
<b>ButtonClick</b> ( <a href="#">see page 82</a> )	Simulates a button click, as if the user had clicked the button.
<b>CloseUp</b> ( <a href="#">see page 82</a> )	Generates an OnCloseUp ( <a href="#">see page 85</a> ) event and makes some other actions.
<b>Create</b> ( <a href="#">see page 82</a> )	Creates and initializes a TBtnEdit instance.
<b>Destroy</b> ( <a href="#">see page 82</a> )	Destroys an instance of TBtnEdit
<b>DoDropDownKeys</b> ( <a href="#">see page 83</a> )	Works as a method dispatcher depending on parameter values.
<b>DropDown</b> ( <a href="#">see page 83</a> )	Generates an OnDropDown ( <a href="#">see page 85</a> ) event.
<b>EndTracking</b> ( <a href="#">see page 83</a> )	Called after finishing the mouse tracking
<b>KeyPress</b> ( <a href="#">see page 83</a> )	Respond to keyboard input.
<b>MouseDown</b> ( <a href="#">see page 83</a> )	Overrides inherited method
<b>MouseMove</b> ( <a href="#">see page 83</a> )	Overrides the inherited method.
<b>PaintBtnGlyph</b> ( <a href="#">see page 84</a> )	Overrides inherited method
<b>StartTracking</b> ( <a href="#">see page 84</a> )	Overrides inherited property

**TCustomPropertyEdit Class**










<b>TCustomPropertyEdit Class</b>	<b>Description</b>
<b>AcceptListValue</b> ( <a href="#">see page 453</a> )	Called when item was selected in drop-down list.
<b>ButtonClick</b> ( <a href="#">see page 453</a> )	Called when button was clicked.
<b>Change</b> ( <a href="#">see page 453</a> )	Reflects changes in property editor to property
<b>ChangePropertyValue</b> ( <a href="#">see page 453</a> )	Writes current text to property.
<b>Create</b> ( <a href="#">see page 453</a> )	Creates and initializes a TCustomPropertyEdit instance.
<b>DbClick</b> ( <a href="#">see page 453</a> )	Calls corresponding property editor
<b>DoEdit</b> ( <a href="#">see page 454</a> )	Calls corresponding property editor
<b>DoExit</b> ( <a href="#">see page 454</a> )	Sets value to corresponding property
<b>DropDown</b> ( <a href="#">see page 454</a> )	Opens drop down list.
<b>GetEditor</b> ( <a href="#">see page 454</a> )	Gets the corresponding editor for the property
<b>GetStr</b> ( <a href="#">see page 454</a> )	Adds string to the pick list of the TCustomPropertyEdit ( <a href="#">see page 448</a> )
<b>GetWStr</b> ( <a href="#">see page 454</a> )	Adds Unicode string to the pick list of the TCustomPropertyEdit ( <a href="#">see page 448</a> )
<b>MouseDown</b> ( <a href="#">see page 454</a> )	Overrides inherited method
<b>Notification</b> ( <a href="#">see page 455</a> )	Responds to notifications that components are being created or destroyed.
<b>PaintStatus</b> ( <a href="#">see page 455</a> )	Paints status area. This method is called only if StatusWidth is greater 0.
<b>SetValue</b> ( <a href="#">see page 455</a> )	Sets Value to the property editor and invokes OnChange if assigned.
<b>UpdateEditState</b> ( <a href="#">see page 455</a> )	Updates all the properties of the TCustomPropertyEdit ( <a href="#">see page 448</a> ) object

**TUnicodeEdit Properties**





<b>TUnicodeEdit Properties</b>	<b>Description</b>
<b>IsUnicode</b> ( <a href="#">see page 108</a> )	Specifies whether control is Unicode edit.

 SelTextW (see page 109)	Specifies the selected portion of the edit control's text (Unicode version).
 Text (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW (see page 109)	Specifies the text string that is displayed in the edit box (Ansi version).








**TBtEdit Class**

TBtEdit Class	Description
 Alignment (see page 76)	Determines how the text is aligned within the editor control.
 ButtonVisible (see page 76)	Specifies if button-like rectangle at the right edge of the control is visible.
 ButtonWidth (see page 77)	Specifies width in pixels of the button.
 Canvas (see page 77)	Provides access to the drawing surface of the TBtEdit (see page 70).
 MultiLine (see page 77)	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth (see page 77)	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns (see page 77)	Determines whether the user can insert return characters into the text.
 WantTabs (see page 78)	Determines whether the user can insert tab characters into the text.
 WordWrap (see page 78)	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.









**TCustomEditEx Class**

TCustomEditEx Class	Description
 ActiveList (see page 84)	Specifies current popup control.
 EditStyle (see page 84)	Specifies edit style of the button
 ListAlign (see page 84)	Specifies relative align of popup control.
 PickList (see page 84)	Determines the drop-down list.

**TCustomPropertyEdit Class**

TCustomPropertyEdit Class	Description
 AcceptTab (see page 455)	Specifies whether edit control should process TAB key.
 Component (see page 456)	Specifies owner of the property will be edited
 Designer (see page 456)	Specifies active designer.
 PropertyEditor (see page 456)	Specifies interface for property editor (IProperty)
 PropertyName (see page 456)	Specifies name of the property which will be edited
 ReadOnly (see page 456)	Specifies whether property can not be edited.
 TypeKinds (see page 456)	Set of types of editable properties.





**TPropertyEdit Class**

TPropertyEdit Class	Description
 Component (see page 462)	Specifies owner of the property will be edited
 Designer (see page 462)	Specifies active designer.
 ListAlign (see page 462)	Specifies relative align of popup control.
 OnSetPropValueA (see page 462)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 463)	Occurs before changing property value (Unicode version).
 PropertyName (see page 463)	Specifies name of the property which will be edited
 ReadOnly (see page 463)	Specifies whether property can not be edited.
 TypeKinds (see page 463)	Set of types of editable properties.

**TBtEdit Events****TBtEdit Class**

TBtEdit Class	Description
 OnButtonClick (see page 78)	Occurs when the user clicks the button.

**TCustomEditEx Class**

TCustomEditEx Class	Description
 OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
 OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
 OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
 OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.



**TCustomPropertyEdit Class**

TCustomPropertyEdit Class	Description
OnSetPropValueA (see page 457)	Occurs before changing property value (ANSI version).
OnSetPropValueW (see page 457)	Occurs before changing property value (Unicode version).

**Legend**

	Constructor
	virtual
	protected
	Property
	read only
	Event

**TBtnEdit Events****TBtnEdit Class**

TBtnEdit Class	Description
OnButtonClick (see page 78)	Occurs when the user clicks the button.

**TCustomEditEx Class**

TCustomEditEx Class	Description
OnAcceptListValue (see page 85)	Occurs when the user makes right choice in the drop-down list.
OnCloseUp (see page 85)	Occurs when the drop-down list closes up due to some user action.
OnDropDown (see page 85)	Occurs when the user opens the drop-down list.
OnMeasureWidth (see page 85)	Occurs when width of controls needs to be calculated.

**TCustomPropertyEdit Class**

TCustomPropertyEdit Class	Description
OnSetPropValueA (see page 457)	Occurs before changing property value (ANSI version).
OnSetPropValueW (see page 457)	Occurs before changing property value (Unicode version).

























**TUnicodeEdit Methods**

TUnicodeEdit Methods	Description
Create (see page 108)	Creates and initializes a TUnicodeEdit instance.
Destroy (see page 108)	Destroys an instance of TUnicodeEdit.




























**TBtnEdit Class**

TBtnEdit Class	Description
AdjustClientRect (see page 73)	Overrides the inherited method.
ButtonClick (see page 73)	Simulates a button click, as if the user had clicked the button.
Create (see page 73)	Creates and initializes a TBtnEdit (see page 70) instance.
CreateParams (see page 73)	Overrides the inherited method.
CreateWnd (see page 73)	Overrides the inherited method.
Destroy (see page 74)	Destroys an instance of TBtnEdit (see page 70)
EndTracking (see page 74)	Called after finishing the mouse tracking
KeyDown (see page 74)	Overrides the inherited method.
KeyPress (see page 74)	Overrides the inherited method.
MouseMove (see page 74)	Overrides the inherited method.
MouseUp (see page 75)	Overrides the inherited method.
Paint (see page 75)	Overrides the base rendering method.
PaintBtnGlyph (see page 75)	Renders the image of the button.
PaintStatus (see page 75)	Paints status area. This method is called only if StatusWidth (see page 77) is greater 0.
PaintWindow (see page 75)	Overrides the inherited method.
PtInButton (see page 75)	Checks if specified point is over the button
StartTracking (see page 76)	Calls immediately after user pushes the button.
StopTracking (see page 76)	Calls immediately after user releases the button.
TrackButton (see page 76)	Controls tracking button process.





**TCustomEditEx Class**

<b>TCustomEditEx Class</b>	<b>Description</b>
  AcceptListValue ( <a href="#">see page 82</a> )	Active pop-up window.
  ButtonClick ( <a href="#">see page 82</a> )	Simulates a button click, as if the user had clicked the button.
 CloseUp ( <a href="#">see page 82</a> )	Generates an OnCloseUp ( <a href="#">see page 85</a> ) event and makes some other actions.
  Create ( <a href="#">see page 82</a> )	Creates and initializes a TBtnEdit instance.
  Destroy ( <a href="#">see page 82</a> )	Destroys an instance of TBtnEdit
  DoDropDownKeys ( <a href="#">see page 83</a> )	Works as a method dispatcher depending on parameter values.
 DropDown ( <a href="#">see page 83</a> )	Generates an OnDropDown ( <a href="#">see page 85</a> ) event.
  EndTracking ( <a href="#">see page 83</a> )	Called after finishing the mouse tracking
  KeyPress ( <a href="#">see page 83</a> )	Respond to keyboard input.
  MouseDown ( <a href="#">see page 83</a> )	Overrides inherited method
  MouseMove ( <a href="#">see page 83</a> )	Overrides the inherited method.
  PaintBtnGlyph ( <a href="#">see page 84</a> )	Overrides inherited method
  StartTracking ( <a href="#">see page 84</a> )	Overrides inherited property











**TCustomPropertyEdit Class**

<b>TCustomPropertyEdit Class</b>	<b>Description</b>
  AcceptListValue ( <a href="#">see page 453</a> )	Called when item was selected in drop-down list.
  ButtonClick ( <a href="#">see page 453</a> )	Called when button was clicked.
  Change ( <a href="#">see page 453</a> )	Reflects changes in property editor to property
  ChangePropertyValue ( <a href="#">see page 453</a> )	Writes current text to property.
  Create ( <a href="#">see page 453</a> )	Creates and initializes a TCustomPropertyEdit instance.
  DbClick ( <a href="#">see page 453</a> )	Calls corresponding property editor
 DoEdit ( <a href="#">see page 454</a> )	Calls corresponding property editor
  DoExit ( <a href="#">see page 454</a> )	Sets value to corresponding property
 DropDown ( <a href="#">see page 454</a> )	Opens drop down list.
 GetEditor ( <a href="#">see page 454</a> )	Gets the corresponding editor for the property
 GetStr ( <a href="#">see page 454</a> )	Adds string to the pick list of the TCustomPropertyEdit ( <a href="#">see page 448</a> )
 GetWStr ( <a href="#">see page 454</a> )	Adds Unicode string to the pick list of the TCustomPropertyEdit ( <a href="#">see page 448</a> )
  MouseDown ( <a href="#">see page 454</a> )	Overrides inherited method
  Notification ( <a href="#">see page 455</a> )	Responds to notifications that components are being created or destroyed.
  PaintStatus ( <a href="#">see page 455</a> )	Paints status area. This method is called only if StatusWidth is greater 0.
 SetValue ( <a href="#">see page 455</a> )	Sets Value to the property editor and invokes OnChange if assigned.
 UpdateEditState ( <a href="#">see page 455</a> )	Updates all the properties of the TCustomPropertyEdit ( <a href="#">see page 448</a> ) object





**TUnicodeEdit Properties**

<b>TUnicodeEdit Properties</b>	<b>Description</b>
 IsUnicode ( <a href="#">see page 108</a> )	Specifies whether control is Unicode edit.
 SelTextW ( <a href="#">see page 109</a> )	Specifies the selected portion of the edit control's text (Unicode version).
 Text ( <a href="#">see page 109</a> )	Specifies the text string that is displayed in the edit box (Ansi version).
 TextW ( <a href="#">see page 109</a> )	Specifies the text string that is displayed in the edit box (Ansi version).








**TBtnEdit Class**

<b>TBtnEdit Class</b>	<b>Description</b>
 Alignment ( <a href="#">see page 76</a> )	Determines how the text is aligned within the editor control.
 ButtonVisible ( <a href="#">see page 76</a> )	Specifies if button-like rectangle at the right edge of the control is visible.
 ButtonWidth ( <a href="#">see page 77</a> )	Specifies width in pixels of the button.
  Canvas ( <a href="#">see page 77</a> )	Provides access to the drawing surface of the TBtnEdit ( <a href="#">see page 70</a> ).
 MultiLine ( <a href="#">see page 77</a> )	Designates a multiline edit control. The default is single-line edit control.
 StatusWidth ( <a href="#">see page 77</a> )	Specifies width of status area. If StatusWidth is equal to 0 - status area is not processed.
 WantReturns ( <a href="#">see page 77</a> )	Determines whether the user can insert return characters into the text.
 WantTabs ( <a href="#">see page 78</a> )	Determines whether the user can insert tab characters into the text.
 WordWrap ( <a href="#">see page 78</a> )	Determines whether the edit control inserts soft carriage returns so text wraps at the right margin.









**TCustomEditEx Class**

TCustomEditEx Class	Description
 ActiveList (see page 84)	Specifies current popup control.
 EditStyle (see page 84)	Specifies edit style of the button
 ListAlign (see page 84)	Specifies relative align of popup control.
 PickList (see page 84)	Determines the drop-down list.

**TCustomPropertyEdit Class**

TCustomPropertyEdit Class	Description
 AcceptTab (see page 455)	Specifies whether edit control should process TAB key.
 Component (see page 456)	Specifies owner of the property will be edited
 Designer (see page 456)	Specifies active designer.
 PropertyEditor (see page 456)	Specifies interface for property editor (IProperty)
 PropertyName (see page 456)	Specifies name of the property which will be edited
 ReadOnly (see page 456)	Specifies whether property can not be edited.
 TypeKinds (see page 456)	Set of types of editable properties.

**TPropertyEdit Class**

TPropertyEdit Class	Description
 Component (see page 462)	Specifies owner of the property will be edited
 Designer (see page 462)	Specifies active designer.
 ListAlign (see page 462)	Specifies relative align of popup control.
 OnSetPropValueA (see page 462)	Occurs before changing property value (ANSI version).
 OnSetPropValueW (see page 463)	Occurs before changing property value (Unicode version).
 PropertyName (see page 463)	Specifies name of the property which will be edited
 ReadOnly (see page 463)	Specifies whether property can not be edited.
 TypeKinds (see page 463)	Set of types of editable properties.

**1.26.1.2.1 TPropertyEdit Properties****1.26.1.2.1.1 TPropertyEdit.Component Property**

Specifies owner of the property will be edited

```
property Component: TComponent;
```

**Description****1.26.1.2.1.2 TPropertyEdit.Designer Property**

Specifies active designer.

```
property Designer: TzFormDesigner;
```

**Description**

This property is used for some property editors demanding transfer IDesigner as parameter

**1.26.1.2.1.3 TPropertyEdit.ListAlign Property**

Specifies relative align of popup control.

```
property ListAlign: TAlignment;
```

**1.26.1.2.1.4 TPropertyEdit.OnSetPropValueA Property**

Occurs before changing property value (ANSI version).

```
property OnSetPropValueA: TOnSetPropValueEventA;
```

**Description**

Write OnSetPropValueA to change value to be written to property.

**1.26.1.2.1.5 TPropertyEdit.OnSetPropValueW Property**

Occurs before changing property value (Unicode version).

```
property OnSetPropValueW: TOnSetPropValueEventW;
```

**Description**

Write OnSetPropValueW to change value to be written to property.

**1.26.1.2.1.6 TPropertyEdit.PropertyName Property**

Specifies name of the property which will be edited

```
property PropertyName: string;
```

**Description****1.26.1.2.1.7 TPropertyEdit.ReadOnly Property**

Specifies whether property can not be edited.

```
property ReadOnly: Boolean;
```

**Description****1.26.1.2.1.8 TPropertyEdit.TypeKinds Property**

Set of types of editable properties.

```
property TypeKinds: TTypeKinds;
```

**Description**

This property used to determine if assigned property editor does will be accessible or not.

If assigned property editor does not suit this TypeKinds, it will be inaccessible.

**1.26.1.3 TPropertyNameProperty Class**

TPropertyNameProperty is the editor for string property with name "PropertyName"

**Class Hierarchy**

```
TPropertyNameProperty = class(TStringProperty);
```

**File**


edcPropEdit

**Description**

# 1.26.2 Interfaces

The following table lists interfaces in this documentation.

## Interfaces

Interface	Description
 IPropertyStatusImage ( <a href="#">see page 464</a> )	Provides status image information.

## Legend

	Interface
-----------------------------------------------------------------------------------	-----------

## 1.26.2.1 IPropertyStatusImage Interface

Provides status image information.

### Class Hierarchy

 edcPropEdit.IPropertyStatusImage
--------------------------------------------------------------------------------------------------------------------

```
IPropertyStatusImage = interface;
```

### File




edcPropEdit

### Description

Add this interface to property editors to support status region in property edit (special area at the left of edit control).

### Members




#### IPropertyStatusImage Methods

IPropertyStatusImage Methods	Description
 DrawStatus ( <a href="#">see page 464</a> )	Draws status image.
 GetStatusWidth ( <a href="#">see page 464</a> )	Returns width of status image.
 StatusClick ( <a href="#">see page 465</a> )	Called when user clicks on status region in edit control.

## Legend

	Method
-------------------------------------------------------------------------------------	--------

#### IPropertyStatusImage Methods

IPropertyStatusImage Methods	Description
 DrawStatus ( <a href="#">see page 464</a> )	Draws status image.
 GetStatusWidth ( <a href="#">see page 464</a> )	Returns width of status image.
 StatusClick ( <a href="#">see page 465</a> )	Called when user clicks on status region in edit control.

### 1.26.2.1.1 IPropertyStatusImage Methods

#### 1.26.2.1.1.1 IPropertyStatusImage.DrawStatus Method

Draws status image.

```
procedure DrawStatus(Canvas: TCanvas; const R: TRect);
```

#### 1.26.2.1.1.2 IPropertyStatusImage.GetStatusWidth Method

Returns width of status image.

```
function GetStatusWidth: integer;
```

### 1.26.2.1.1.3 IPropertyStatusImage.StatusClick Method

Called when user clicks on status region in edit control.

```
procedure StatusClick(Button: TMouseButton; pt: TPoint);
```

## 1.26.3 Types

The following table lists types in this documentation.

### Types

Type	Description
TOnSetPropValueEventA (🔗 see page 465)	See TCustomPropertyEdit.OnSetPropValueA Event (🔗 see page 457)
TOnSetPropValueEventW (🔗 see page 465)	See TCustomPropertyEdit.OnSetPropValueW Event (🔗 see page 457)

### 1.26.3.1 edcPropEdit.TOnSetPropValueEventA Type

See TCustomPropertyEdit.OnSetPropValueA Event (🔗 see page 457)

```
TOnSetPropValueEventA = procedure (Sender: TObject; var Value: string) of object;
```

**File**

edcPropEdit

### 1.26.3.2 edcPropEdit.TOnSetPropValueEventW Type

See TCustomPropertyEdit.OnSetPropValueW Event (🔗 see page 457)

```
TOnSetPropValueEventW = procedure (Sender: TObject; var Value: WideString) of object;
```

**File**

edcPropEdit

## 1.27 eddAlignDlg Namespace

### 1.27.1 Classes

The following table lists classes in this documentation.

### Classes

Class	Description
TAlignmentDlg (🔗 see page 465)	Alignment dialog box

### 1.27.1.1 TAlignmentDlg Class

Alignment dialog box

Class Hierarchy



```
TAlignmentDlg = class(TForm);
```

File

eddAlignDlg

Description

Use this dialog box to line up selected components in relation to each other or to the form. The options for horizontal or vertical alignment are:

Option	Description
No change	Does not change the alignment of the component
Left sides	Lines up the left edges of the selected components (horizontal only)
Centers	Lines up the centers of the selected components
Right sides	Lines up the right edges of the selected components (horizontal only)
Tops	Lines up the top edges of the selected components (vertical only)
Bottoms	Lines up the bottom edges of the selected components (vertical only)
Space equally	Lines up the selected components equidistant from each other
Center in window	Lines up the selected components with the center of the window

You can also invoke Align by right clicking in an active form.

# 1.28 eddAlignPal Namespace

## 1.28.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TAlignPalette (see page 466)	Alignment palette

### 1.28.1.1 TAlignPalette Class

Alignment palette

Class Hierarchy



```
TAlignPalette = class(TForm);
```

File

eddAlignPal

Description

Use the alignment palette to align components to the form, or with each other.

The alignment palette has Tool Help for each button.

Icon	Effect
	Aligns the selected components to the left edge of the component first selected. (Not applicable for single components.)
	Moves the selected components horizontally until their centers are aligned with the component first selected. (Not applicable for single components.)
	Aligns the selected component(s) to the center of the form along a horizontal line.
	Horizontally aligns three or more selected components so that the middle components are equidistantly spaced between the outer components
	Aligns the selected components to the right edge of the component first selected. (Not applicable for single components.)
	Aligns the selected components to the top edge of the component first selected. (Not applicable for single components.)
	Moves the selected components vertically until their centers are aligned with component first selected. (Not applicable for single components.)
	Aligns the selected component(s) to the center of the form along a vertical line.
	Vertically aligns three or more selected components so that the middle components are equally spaced between the outer components.
	Aligns the selected components to the bottom edge of the component first selected. (Not applicable for single components.)

If you are unsure of how a particular button on the alignment palette acts, click and hold on to the button. The icon on the button changes to show you how it will align the selected components. To apply the button's alignment to a selection, release the button. To prevent alignment after you click and hold the button, drag the mouse off the palette before releasing the mouse button.

Note: You can also use the Alignment dialog box to align components.

# 1.29 eddCrOrdDI Namespace

## 1.29.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TCreateOrderDlg (see page 468)	Creation Order dialog box



### 1.29.1.1 TCreateOrderDlg Class

Creation Order dialog box

**Class Hierarchy**



```
TCreateOrderDlg = class(TForm);
```

**File**

eddCrOrdDlg

**Description**

Use this dialog box to specify the order in which your application creates nonvisual components when you load the form at design time or runtime.

The list box displays only those nonvisual components on the active form, their type, and their current creation order. The default creation order is determined by the order in which you placed the nonvisual components on the form.

To change the creation order:

1. Select a component name.
2. Click the up button to move the component creation order up, or click the down arrow to move its creation order down. You can also drag the selected component to its new position in the creation order.
3. To save your changes, click OK.

## 1.30 eddCustomPal Namespace

### 1.30.1 Classes

The following table lists classes in this documentation.

**Classes**

Class	Description
TCustomizePaletteDlg (see page 468)	Customizing the way the Component palette appears

### 1.30.1.1 TCustomizePaletteDlg Class

Customizing the way the Component palette appears

**Class Hierarchy**



```
TCustomizePaletteDlg = class(TForm);
```

**File**

eddcustomPal

**Description**

Use the Palette page of the Environment Options dialog box to customize the way the Component palette appears. You can rename, add, remove, or reorder pages and components.

- Pages** Lists the pages in the Component palette, in the order in which they currently appear. You can rearrange these pages or view and rearrange their components in the Components list. The last item in the Pages list is [All]; when you select [All], the Components list shows components from every page as well as hidden components.
- Components** Lists the components on the currently selected Component palette page in the Pages list. Components may come from installed packages or they may be component templates created with the Component|Create Component Template command. Components appear in their current order on the palette. You can rearrange components, or move them to a different page by dragging them. When [All] is selected in the Pages list, you can sort by component name, package, or palette page by clicking on the appropriate column heading

Use the following buttons when an item is selected in the Pages list.

- Add** Click Add to display the Add Page dialog box, where you can create new pages on the Component palette. Once you have created a new Component palette page, you can move components from other pages onto it or add new components onto it using Component|Install
- Delete** To remove the selected page from the palette, click Delete. Before you can delete a page, it must be empty of components. If you accidentally delete a component, select [All] in the Pages list and press Default Pages, or use Component|Install to add it
- Rename** Click Rename to display the Rename Page dialog box, where you can rename the selected page
- Default Pages** This button is available when [All] is selected on the Pages list. Click Default Pages to restore pages to their default order and replace all components on their default pages
- Move Up / Move Down** To change the position of the selected page, click Move Up or Move Down. You can also drag pages to a new position

Use the following buttons when an item is selected in the Components list.

- Hide Show** / To prevent an installed component from appearing on the Component palette, click Hide. To redisplay a hidden component, select [All] on the Pages list, select the hidden component on the Components list, then click Show
- Delete** This button is available only when a component template is selected. To delete a component template, click Delete
- Move Up / Move Down** / To change the position of a component on a page, click Move Up or Move Down. You can also drag components to a new position

---

## 1.31 edcToolList Namespace

## 1.31.1 Classes

The following table lists classes in this documentation.

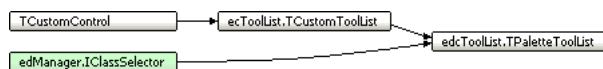
### Classes

Class	Description
TPaletteToolList (see page 470)	Represents component palette in BDS style.

### 1.31.1.1 TPaletteToolList Class

Represents component palette in BDS style.

#### Class Hierarchy



```
TPaletteToolList = class(TCustomToolList, IClassSelector);
```

#### File

edcToolList

#### Description

Tool list contains components from component palette. Component page is represented by the tool list category. Items (see page 482) may be easily adjusted by user using drag&drop operations.

#### Members

##### TCustomToolList Methods









TCustomToolList Methods	Description
CollapseAll (see page 118)	Collapses all categories.
Create (see page 119)	Creates and initializes a TCustomToolList instance.
Destroy (see page 119)	Destroys an instance of TCustomToolList.
DrawItemImage (see page 119)	Draws item image.
ExpandAll (see page 119)	Expands all categories.
GetCategoryItem (see page 119)	Returns index of category item at the given position or above.
ItemAtPos (see page 119)	Returns items at the given position. If there are no item at the specified position function returns -1.
ItemIndexChanged (see page 119)	Called when selected item was changed.
ItemRect (see page 119)	Returns rectangle occupied by the item.
ItemsArranged (see page 120)	Called after items were rearranged by the drag&drop operations.
ItemsChanged (see page 120)	Called when items were changed (any changes).
ItemsHeight (see page 120)	Calculates total height of items.
MakeTopItem (see page 120)	Scrolls list to make specified item topmost item.
MakeVisible (see page 120)	Scrolls list to make specified item visible.
PaintItem (see page 120)	Calls TToolListItem.Paint and allows to customize item rendering in derived classes.
SelectFirstVisible (see page 120)	Selects first visible, i.e. not hidden, item.

##### IClassSelector Interface




















IClassSelector Interface	Description
ClsChanged (see page 511)	Called when selected in component palette component class was changed.
ClsPalChanged (see page 512)	Called when component palette was changed.

##### TPaletteToolList Class














TPaletteToolList Class	Description
ClsChanged (see page 476)	Called when selected in component palette component class was changed.






































 CIsPalChanged (see page 476)	Called when component palette was changed.
 ComponentAt (see page 476)	Returns associated component class info for toll item at Index position.
 Create (see page 476)	Creates and initializes a TPaletteToolList instance.
 Destroy (see page 477)	Destroys an instance of TPaletteToolList.
 DrawItemImage (see page 477)	Draws item image.
 ItemIndexChanged (see page 477)	Called when selected item was changed.
 ItemsArranged (see page 477)	Called after items were rearranged by the drag&drop operations.
 ShowCategory (see page 477)	Scrolls tool list to make category visible.





























## TCustomToolList Properties

TCustomToolList Properties	Description
 AllowArrange (see page 120)	Specifies whether items can be arranged by the drag&drop operations.
 AutoCollapse (see page 120)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
 CategoryHeight (see page 120)	Specifies height of category item.
 Filtered (see page 120)	Specifies whether items are filtered.
 FilterString (see page 121)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 121)	Holds folding icon images.
 HintProps (see page 121)	Provide properties to adjust hints processing.
 Images (see page 121)	Determines which image list is associated with the tool list.
 InsertAtItem (see page 121)	Specifies item index at which dragged object can be dropped.
 ItemHeight (see page 121)	Specifies height of the normal item.
 ItemIndex (see page 121)	Indicates which item is selected. If no item is selected ItemIndex is equal to -1.
 Items (see page 121)	Provides access to items displayed in tool list.
 MouseOverItem (see page 122)	Indicates item over which mouse cursor is located.
 RightClickSelect (see page 122)	Specifies whether item can be selected by mouse right click.
 RowSpace (see page 122)	Specifies space between two sequential items.
 Selected (see page 122)	Currently selected item in tool list.
 StyleCategory (see page 122)	Specifies style of category items.
 StyleCategoryMouseOver (see page 122)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (see page 122)	Specifies style of selected category item.
 StyleItem (see page 122)	Specifies style of tool items.
 StyleItemMouseOver (see page 122)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (see page 122)	Specifies style of selected tool item.
 VerticalGroups (see page 122)	Specifies whether category item should be displayed vertically along owned items.
 ViewOrigin (see page 123)	Specifies scrolling position of the tool list control.





## TPaletteToolList Class

TPaletteToolList Class	Description
 Align (see page 477)	Determines how the control aligns within its container (parent control).
 AllowArrange (see page 478)	Specifies whether items can be arranged by the drag&drop operations.
 Anchors (see page 478)	Specifies how the control is anchored to its parent.
 AutoCollapse (see page 478)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
 BevelEdges (see page 478)	Specifies which edges of the control are beveled.
 BevelInner (see page 478)	Specifies the cut of the inner bevel.
 BevelKind (see page 479)	Specifies the control's bevel style.
 BevelOuter (see page 479)	Specifies the cut of the outer bevel.
 BiDiMode (see page 479)	Specifies the bi-directional mode for the control.
 CategoryHeight (see page 479)	Specifies height of category item.
 Color (see page 479)	Specifies the background color of the control.
 Constraints (see page 480)	Specifies the size constraints for the control.
 Ctl3D (see page 480)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

 CustomItems (see page 480)	<p>Allows managing of items in component toll list. Item in list means:</p> <ol style="list-style-type: none"> <li>1. Specification component item ComponentClassName</li> <li>2. Specification new category +CategoryName</li> <li>3. Specification existed category (components page) with adding all components which belong to this page. ++CategoryName</li> <li>4. Rename existed category (components page) with adding all components which belong to this page. ++CategoryName=Display_name_of_category</li> <li>5. Adding all components which belong to the page without adding category. ++CategoryName=</li> </ol>
 DragCursor (see page 481)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragImageType (see page 481)	Specifies drag image when dragging component on form.
 DragKind (see page 481)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 481)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 481)	Controls whether the control responds to mouse, keyboard, and timer events.
 Filtered (see page 481)	Specifies whether items are filtered.
 FilterString (see page 482)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 482)	Holds folding icon images.
 Font (see page 482)	Controls the attributes of text written on or in the control.
 HintProps (see page 482)	Provide properties to adjust hints processing.
 ItemHeight (see page 482)	Specifies height of the normal item.
 Items (see page 482)	Provides access to items displayed in tool list.
 OnCanResize (see page 482)	Occurs when an attempt is made to resize the control.
 OnClick (see page 483)	Occurs when the user clicks the control.
 OnConstrainedResize (see page 483)	Adjust resize constraints.
 OnContextPopup (see page 483)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 484)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 484)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 484)	Occurs when the user drags an object over a control.
 OnEndDrag (see page 484)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 484)	Occurs when a control receives the input focus.
 OnExit (see page 485)	Occurs when the input focus shifts away from one control to another.
 OnKeyDown (see page 485)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 485)	Occurs when key pressed.
 OnKeyUp (see page 485)	Occurs when the user releases a key that has been pressed.
 OnMouseDown (see page 486)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 486)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 486)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.
 OnResize (see page 486)	Occurs immediately after the control is resized.
 OnStartDrag (see page 487)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 487)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 487)	Determines where a control looks for its color information.
 ParentCtl3D (see page 487)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 487)	Determines where a control looks for its font information.
 ParentShowHint (see page 488)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 488)	Identifies the pop-up menu associated with the control.

 RowSpace (  see page 488)	Specifies space between two sequential items.
 ShowCaptions (  see page 488)	Specifies whether button captions displayed in tool list.
 ShowHint (  see page 488)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 StyleCategory (  see page 488)	Specifies style of category items.
 StyleCategoryMouseOver (  see page 488)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (  see page 488)	Specifies style of selected category item.
 StyleItem (  see page 488)	Specifies style of tool items.
 StyleItemMouseOver (  see page 488)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (  see page 488)	Specifies style of selected tool item.
 TabOrder (  see page 489)	Indicates the position of the control in its parent's tab order.
 TabStop (  see page 489)	Determines if the user can tab to a control.
 TransparentImages (  see page 489)	Specifies whether bottom-left pixel is transparent color. If TransparentImages is False, images are drawn without transparency.
 VerticalGroups (  see page 489)	Specifies whether category item should be displayed vertically along owned items.
 Visible (  see page 489)	Determines whether the component appears on screen.








## TCustomToolList Events

TCustomToolList Events	Description
 OnItemArranged (  see page 123)	Occurs when items order was changed.
 OnItemChanged (  see page 123)	Occurs when selected item is changes.


## TPaletteToolList Class

TPaletteToolList Class	Description
 OnPalChange (  see page 489)	Occurs when component palette was changed.


## Legend

	Method
	virtual
	protected
	private
	Property
	read only
	Event

























## TCustomToolList Events





TCustomToolList Events	Description
 OnItemArranged (  see page 123)	Occurs when items order was changed.
 OnItemChanged (  see page 123)	Occurs when selected item is changes.

## TPaletteToolList Class



TPaletteToolList Class	Description
 OnPalChange (  see page 489)	Occurs when component palette was changed.

## TCustomToolList Methods










TCustomToolList Methods	Description
 CollapseAll (  see page 118)	Collapses all categories.
 Create (  see page 119)	Creates and initializes a TCustomToolList instance.
 Destroy (  see page 119)	Destroys an instance of TCustomToolList.
 DrawItemImage (  see page 119)	Draws item image.
 ExpandAll (  see page 119)	Expands all categories.
 GetCategoryItem (  see page 119)	Returns index of category item at the given position or above.
 ItemAtPos (  see page 119)	Returns items at the given position. If there are no item at the specified position function returns -1.
 ItemIndexChanged (  see page 119)	Called when selected item was changed.
 ItemRect (  see page 119)	Returns rectangle occupied by the item.
 ItemsArranged (  see page 120)	Called after items were rearranged by the drag&drop operations.
 ItemsChanged (  see page 120)	Called when items were changed (any changes).
 ItemsHeight (  see page 120)	Calculates total height of items.

 MakeTopItem (see page 120)	Scrolls list to make specified item topmost item.
 MakeVisible (see page 120)	Scrolls list to make specified item visible.
 PaintItem (see page 120)	Calls TToolListItem.Paint and allows to customize item rendering in derived classes.
 SelectFirstVisible (see page 120)	Selects first visible, i.e. not hidden, item.

























**IClassSelector Interface**

IClassSelector Interface	Description
 ClsChanged (see page 511)	Called when selected in component palette component class was changed.
 ClsPalChanged (see page 512)	Called when component palette was changed.





**TPaletteToolList Class**






































TPaletteToolList Class	Description
 ClsChanged (see page 476)	Called when selected in component palette component class was changed.
 ClsPalChanged (see page 476)	Called when component palette was changed.
 ComponentAt (see page 476)	Returns associated component class info for toll item at Index position.
 Create (see page 476)	Creates and initializes a TPaletteToolList instance.
 Destroy (see page 477)	Destroys an instance of TPaletteToolList.
 DrawItemImage (see page 477)	Draws item image.
 ItemIndexChanged (see page 477)	Called when selected item was changed.
 ItemsArranged (see page 477)	Called after items were rearranged by the drag&drop operations.
 ShowCategory (see page 477)	Scrolls tool list to make category visible.

**TCustomToolList Properties**























TCustomToolList Properties	Description
 AllowArrange (see page 120)	Specifies whether items can be arranged by the drag&drop operations.
 AutoCollapse (see page 120)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
 CategoryHeight (see page 120)	Specifies height of category item.
 Filtered (see page 120)	Specifies whether items are filtered.
 FilterString (see page 121)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 121)	Holds folding icon images.
 HintProps (see page 121)	Provide properties to adjust hints processing.
 Images (see page 121)	Determines which image list is associated with the tool list.
 InsertAtItem (see page 121)	Specifies item index at which dragged object can be dropped.
 ItemHeight (see page 121)	Specifies height of the normal item.
 ItemIndex (see page 121)	Indicates which item is selected. If no item is selected ItemIndex is equal to -1.
 Items (see page 121)	Provides access to items displayed in tool list.
 MouseOverItem (see page 122)	Indicates item over which mouse cursor is located.
 RightClickSelect (see page 122)	Specifies whether item can be selected by mouse right click.
 RowSpace (see page 122)	Specifies space between two sequential items.
 Selected (see page 122)	Currently selected item in tool list.
 StyleCategory (see page 122)	Specifies style of category items.
 StyleCategoryMouseOver (see page 122)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (see page 122)	Specifies style of selected category item.
 StyleItem (see page 122)	Specifies style of tool items.
 StyleItemMouseOver (see page 122)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (see page 122)	Specifies style of selected tool item.
 VerticalGroups (see page 122)	Specifies whether category item should be displayed vertically along owned items.
 ViewOrigin (see page 123)	Specifies scrolling position of the tool list control.

**TPaletteToolList Class**

TPaletteToolList Class	Description
 Align (see page 477)	Determines how the control aligns within its container (parent control).
 AllowArrange (see page 478)	Specifies whether items can be arranged by the drag&drop operations.
 Anchors (see page 478)	Specifies how the control is anchored to its parent.
 AutoCollapse (see page 478)	Specifies whether all categories should be collapsed when one category is expanded or collapsed.
 BevelEdges (see page 478)	Specifies which edges of the control are beveled.

 BevelInner (see page 478)	Specifies the cut of the inner bevel.
 BevelKind (see page 479)	Specifies the control's bevel style.
 BevelOuter (see page 479)	Specifies the cut of the outer bevel.
 BiDiMode (see page 479)	Specifies the bi-directional mode for the control.
 CategoryHeight (see page 479)	Specifies height of category item.
 Color (see page 479)	Specifies the background color of the control.
 Constraints (see page 480)	Specifies the size constraints for the control.
 Ctl3D (see page 480)	Determines whether a control has a three-dimensional (3-D) or two-dimensional look.
 CustomItems (see page 480)	Allows managing of items in component toll list. Item in list means:  1. Specification component item ComponentClassName  2. Specification new category +CategoryName  3. Specification existed category (components page) with adding all components which belong to this page. ++CategoryName  4. Rename existed category (components page) with adding all components which belong to this page. ++CategoryName=Display_name_of_category  5. Adding all components which belong to the page without adding category. ++CategoryName=
 DragCursor (see page 481)	Indicates the image used to represent the mouse pointer when the control is being dragged.
 DragImageType (see page 481)	Specifies drag image when dragging component on form.
 DragKind (see page 481)	Specifies whether the control is being dragged normally or for docking.
 DragMode (see page 481)	Determines how the control initiates drag-and-drop or drag-and-dock operations.
 Enabled (see page 481)	Controls whether the control responds to mouse, keyboard, and timer events.
 Filtered (see page 481)	Specifies whether items are filtered.
 FilterString (see page 482)	Specifies filter string which is used to test item Caption.
 FoldingIcon (see page 482)	Holds folding icon images.
 Font (see page 482)	Controls the attributes of text written on or in the control.
 HintProps (see page 482)	Provide properties to adjust hints processing.
 ItemHeight (see page 482)	Specifies height of the normal item.
 Items (see page 482)	Provides access to items displayed in tool list.
 OnCanResize (see page 482)	Occurs when an attempt is made to resize the control.
 OnClick (see page 483)	Occurs when the user clicks the control.
 OnConstrainedResize (see page 483)	Adjust resize constraints.
 OnContextPopup (see page 483)	Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).
 OnDbClick (see page 484)	Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.
 OnDragDrop (see page 484)	Occurs when the user drops an object being dragged.
 OnDragOver (see page 484)	Occurs when the user drags an object over a control.
 OnEndDrag (see page 484)	Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.
 OnEnter (see page 484)	Occurs when a control receives the input focus.
 OnExit (see page 485)	Occurs when the input focus shifts away from one control to another.
 OnKeyDown (see page 485)	Occurs when a user presses any key while the control has focus.
 OnKeyPress (see page 485)	Occurs when key pressed.
 OnKeyUp (see page 485)	Occurs when the user releases a key that has been pressed.
 OnMouseDown (see page 486)	Occurs when the user presses a mouse button with the mouse pointer over a control.
 OnMouseMove (see page 486)	Occurs when the user moves the mouse pointer while the mouse pointer is over a control.
 OnMouseUp (see page 486)	Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.



 OnResize (see page 486)	Occurs immediately after the control is resized.
 OnStartDrag (see page 487)	Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.
 ParentBiDiMode (see page 487)	Specifies whether the control uses its parent's BiDiMode.
 ParentColor (see page 487)	Determines where a control looks for its color information.
 ParentCtl3D (see page 487)	Determines where a component looks to determine if it should appear three dimensional.
 ParentFont (see page 487)	Determines where a control looks for its font information.
 ParentShowHint (see page 488)	Determines where a control looks to find out if its Help Hint should be shown.
 PopupMenu (see page 488)	Identifies the pop-up menu associated with the control.
 RowSpace (see page 488)	Specifies space between two sequential items.
 ShowCaptions (see page 488)	Specifies whether button captions displayed in tool list.
 ShowHint (see page 488)	Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.
 StyleCategory (see page 488)	Specifies style of category items.
 StyleCategoryMouseOver (see page 488)	Specifies style of category item when mouse is over it.
 StyleCategorySelected (see page 488)	Specifies style of selected category item.
 StyleItem (see page 488)	Specifies style of tool items.
 StyleItemMouseOver (see page 488)	Specifies style of tool item when mouse is over it.
 StyleItemSelected (see page 488)	Specifies style of selected tool item.
 TabOrder (see page 489)	Indicates the position of the control in its parent's tab order.
 TabStop (see page 489)	Determines if the user can tab to a control.
 TransparentImages (see page 489)	Specifies whether bottom-left pixel is transparent color. If TransparentImages is False, images are drawn without transparency.
 VerticalGroups (see page 489)	Specifies whether category item should be displayed vertically along owned items.
 Visible (see page 489)	Determines whether the component appears on screen.

### 1.31.1.1.1 TPaletteToolList Methods

#### 1.31.1.1.1.1 TPaletteToolList.ClsChanged Method

Called when selected in component palette component class was changed.

```
procedure ClsChanged;
```

#### 1.31.1.1.1.2 TPaletteToolList.ClsPalChanged Method

Called when component palette was changed.

```
procedure ClsPalChanged;
```

#### 1.31.1.1.1.3 TPaletteToolList.ComponentAt Method

Returns associated component class info for toll item at Index position.

```
function ComponentAt(Index: integer): TComponentClassInfo;
```

#### 1.31.1.1.1.4 TPaletteToolList.Create Constructor

Creates and initializes a TPaletteToolList instance.

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

#### Description

Use Create to programmatically instantiate a TPaletteToolList component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

### 1.31.1.1.1.5 TPaletteToolList.Destroy Destructor

Destroys an instance of TPaletteToolList.

```
destructor Destroy; override;
```

#### Description

Do not call Destroy directly. Call Free instead. Free verifies that the object reference is not **nil** before calling Destroy.

### 1.31.1.1.1.6 TPaletteToolList.DrawItemImage Method

Draws item image.

```
procedure DrawItemImage(Item: TToolListItem; Canvas: TCanvas; var R: TRect; State: TToolItemState); override;
```

#### Description

By default, it draws image from image list of the tool list, but in derived classes it maybe redefined to get images from another source.

### 1.31.1.1.1.7 TPaletteToolList.ItemIndexChanged Method

Called when selected item was changed.

```
procedure ItemIndexChanged; override;
```

### 1.31.1.1.1.8 TPaletteToolList.ItemsArranged Method

Called after items were rearranged by the drag&drop operations.

```
procedure ItemsArranged; override;
```

### 1.31.1.1.1.9 TPaletteToolList.ShowCategory Method

Scrolls tool list to make category visible.

```
function ShowCategory(const CategoryName: WideString): Boolean;
```

## 1.31.1.1.2 TPaletteToolList Properties

### 1.31.1.1.2.1 TPaletteToolList.Align Property

Determines how the control aligns within its container (parent control).

```
property Align;
```

#### Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to **alLeft**. The value of **alLeft** for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is **alNone**, which means a control remains where it is positioned on a form or panel.

**Tip:** If Align is set to **alClient**, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. The child controls stack up in z-order. To adjust the order in which the controls stack up, drag the controls into their desired positions.

**Note:** To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

#### 1.31.1.1.2.2 TPaletteToolList.AllowArrange Property

Specifies whether items can be arranged by the drag&drop operations.

**property** AllowArrange: Boolean;

#### 1.31.1.1.2.3 TPaletteToolList.Anchors Property

Specifies how the control is anchored to its parent.

**property** Anchors;

##### Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

**Note:** If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

#### 1.31.1.1.2.4 TPaletteToolList.AutoCollapse Property

Specifies whether all categories should be collapsed when one category is expanded or collapsed.

**property** AutoCollapse: Boolean;

#### 1.31.1.1.2.5 TPaletteToolList.BevelEdges Property

Specifies which edges of the control are beveled.

**property** BevelEdges;

##### Description

Use BevelEdges to get or set which edges of the control are beveled. The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

#### 1.31.1.1.2.6 TPaletteToolList.BevelInner Property

Specifies the cut of the inner bevel.

**property** BevelInner;

**Description**

Use BevelInner to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (BevelOuter is bvNone), the inner bevel appears immediately inside the border.

**1.31.1.1.2.7 TPaletteToolList.BevelKind Property**

Specifies the control's bevel style.

```
property BevelKind;
```

**Description**

Use BevelKind to modify the appearance of a bevel. BevelKind influences how sharply the bevel stands out.

BevelKind, in combination with BevelWidth and the cut of the bevel specified by BevelInner or BevelOuter, can create a variety of effects. Experiment with various combinations to get the look you want.

**1.31.1.1.2.8 TPaletteToolList.BevelOuter Property**

Specifies the cut of the outer bevel.

```
property BevelOuter;
```

**Description**

Use BevelInner to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

**1.31.1.1.2.9 TPaletteToolList.BiDiMode Property**

Specifies the bi-directional mode for the control.

```
property BiDiMode;
```

**Description**

Use BiDiMode to enable the control to adjust its appearance and behavior automatically when the application runs in a locale that reads from right to left instead of left to right. The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Alignment does not change for controls that are known to contain number, date, time, or currency values. For example, with data aware controls, the alignment does not change for the following field types: ftSmallint, ftInteger, ftWord, ftFloat, ftCurrency, ftBCD, ftDate, ftTime, ftDateTime, ftAutoInc.

**1.31.1.1.2.10 TPaletteToolList.CategoryHeight Property**

Specifies height of category item.

```
property CategoryHeight: integer;
```

**1.31.1.1.2.11 TPaletteToolList.Color Property**

Specifies the background color of the control.

```
property Color;
```

**Description**

Use Color to read or change the background color of the control.

If a control's `ParentColor` property is true, then changing the `Color` property of the control's parent automatically changes the `Color` property of the control. When the value of the `Color` property is changed, the control's `ParentColor` property is automatically set to false.

#### 1.31.1.1.2.12 `TPaletteToolList.Constraints` Property

Specifies the size constraints for the control.

**property** `Constraints;`

##### Description

Use `Constraints` to specify the minimum and maximum width and height of the control. When `Constraints` contains maximum or minimum values, the control cannot be resized to violate those constraints.

**Warning:** Do not set up constraints that conflict with the value of the `Align` or `Anchors` property. When these properties conflict, the response of the control to resize attempts is not well-defined.

#### 1.31.1.1.2.13 `TPaletteToolList.Ctl3D` Property

Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

**property** `Ctl3D;`

##### Description

`Ctl3D` is provided for backward compatibility. It is not used by 32-bit versions of Windows or NT4.0 and later.

On earlier platforms, `Ctl3D` controlled whether the control had a flat or beveled appearance.

#### 1.31.1.1.2.14 `TPaletteToolList.CustomItems` Property

Allows managing of items in component toll list.

Item in list means:

1. Specification component item

`ComponentClassName`

2. Specification new category

`+CategoryName`

3. Specification existed category (components page) with adding all components which belong to this page.

`++CategoryName`

4. Rename existed category (components page) with adding all components which belong to this page.

`++CategoryName=Display_name_of_category`

5. Adding all components which belong to the page without adding category.

`++CategoryName=`

**property** `CustomItems: TStrings;`

### 1.31.1.1.2.15 TPaletteToolList.DragCursor Property

Indicates the image used to represent the mouse pointer when the control is being dragged.

**property** DragCursor;

#### Description

Use the DragCursor property to change the cursor image presented when the control is being dragged.

**Note:** To make a custom cursor available for the DragCursor property, see the Cursor property.

### 1.31.1.1.2.16 TPaletteToolList.DragImageType Property

Specifies drag image when dragging component on form.

**property** DragImageType: TComponentClassDragImage;

#### Remarks

In Delphi 5,6,7 you will need to add csDisplayDragImage to designed form's ControlStyle.

In Delphi 2005,2006,2007,2009 - DragObject.AlwaysShowDragImages := True, so dragged image is shown over any control.

### 1.31.1.1.2.17 TPaletteToolList.DragKind Property

Specifies whether the control is being dragged normally or for docking.

**property** DragKind;

#### Description

Use DragKind to get or set whether the control participates in drag-and-drop operations, or drag-and-dock operations.

### 1.31.1.1.2.18 TPaletteToolList.DragMode Property

Determines how the control initiates drag-and-drop or drag-and-dock operations.

**property** DragMode;

#### Description

Use DragMode to control when the user can drag the control. Disable the drag-and-drop or drag-and-dock capability at runtime by setting the DragMode property value to dmManual. Enable automatic dragging by setting DragMode to dmAutomatic.

### 1.31.1.1.2.19 TPaletteToolList.Enabled Property

Controls whether the control responds to mouse, keyboard, and timer events.

**property** Enabled;

#### Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

### 1.31.1.1.2.20 TPaletteToolList.Filtered Property

Specifies whether items are filtered.

**property** Filtered: Boolean;

#### Description

Use Filtered property to toggle items filtration. Items are filtered using FilterString property.

#### 1.31.1.1.2.21 TPaletteToolList.FilterString Property

Specifies filter string which is used to test item Caption.

```
property FilterString: string;
```

#### 1.31.1.1.2.22 TPaletteToolList.FoldingIcon Property

Holds folding icon images.

```
property FoldingIcon: TBitmap;
```

##### Description

FoldingIcon should contain two images in a row, first - collapse icon (-), second - expand icon (+).

Color of bottom-left pixel is used as mask color.

Folding icon is initialized from resource when control is created at design time.

#### 1.31.1.1.2.23 TPaletteToolList.Font Property

Controls the attributes of text written on or in the control.

```
property Font;
```

##### Description

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

#### 1.31.1.1.2.24 TPaletteToolList.HintProps Property

Provide properties to adjust hints processing.

```
property HintProps: TechHintHelper;
```

#### 1.31.1.1.2.25 TPaletteToolList.ItemHeight Property

Specifies height of the normal item.

```
property ItemHeight: integer;
```

#### 1.31.1.1.2.26 TPaletteToolList.Items Property

Provides access to items displayed in tool list.

```
property Items: TToolListItems;
```

##### Description

Read Items to access the list of items that appears in the tool list. Use the methods of Items to add, insert, delete and move items.

#### 1.31.1.1.2.27 TPaletteToolList.OnCanResize Property

Occurs when an attempt is made to resize the control.

```
property OnCanResize;
```

##### Description

Use OnCanResize to adjust the way a control is resized. If necessary, change the new width and height of the control in the OnCanResize event handler. The OnCanResize event handler also allows applications to indicate that the entire resize should be aborted.

If there is no OnCanResize event handler, or if the OnCanResize event handler indicates that the resize attempt can

proceed, the OnCanResize event is followed immediately by an OnConstrainedResize event.

#### 1.31.1.1.2.28 TPaletteToolList.OnClick Property

Occurs when the user clicks the control.

**property** OnClick;

#### 1.31.1.1.2.29 TPaletteToolList.OnConstrainedResize Property

Adjust resize constraints.

**property** OnConstrainedResize;

##### Description

Use OnConstrainedResize to adjust a control's constraints when an attempt is made to resize it. Upon entry to the OnConstrainedResize event handler, the parameters of the event handler are set to the corresponding properties of the control's Constraints object. The event handler can adjust those values before they are applied to the new height and width that is being applied to the control. (The CanAutoSize method or an OnCanResize event handler may already have adjusted this new height and width).

On exit from the OnConstrainedResize event handler, the constraints are applied to the attempted new height and width. Once the constraints are applied, the control's height and width are changed. After the control's height and width change, an OnResize event occurs to allow any final adjustments or responses.

##### Notes

The OnConstrainedResize handler is called immediately after the OnCanResize handler.

#### 1.31.1.1.2.30 TPaletteToolList.OnContextPopup Property

Occurs when the user right-clicks the control or otherwise invokes the popup menu (such as using the keyboard).

**property** OnContextPopup;

##### Description

The OnContextPopup handler is called when the user uses the mouse or keyboard to request a popup menu. The OnContextPopup event is generated by a WM\_CONTEXTMENU message, which is itself generated by the user clicking the right mouse button or by pressing SHIFT+F10 or the Applications key.

This event is especially useful when the control does not have an associated popup menu (the PopupMenu property is not set) or if the AutoPopup property of the control's associated popup menu is false. However, the OnContextPopup can also be used to override the automatic context menu that appears when the control has an associated popup menu with an AutoPopup property of true. In this last case, if the event handler displays its own menu, it should set the Handled parameter to true to suppress the default context menu.

The handler's MousePos parameter indicates the position of the mouse, in client coordinates.. If the event was not generated by a mouse click, MousePos is (-1,-1).

**Note:** Parent controls receive an OnContextPopup event before their child controls. In addition, for many child controls, the default window procedure causes the parent control to receive an OnContextPopup event after the child control. As a result, when parent controls do not set Handled to true in an OnContextPopup event handler, the event handler may be called multiple times for each context menu invocation.



### 1.31.1.1.2.31 TPaletteToolList.OnDbClick Property

Occurs when the user double-clicks the left mouse button when the mouse pointer is over the control.

```
property OnDbClick;
```

#### Description

Use the OnDbClick event to respond to mouse double-clicks.

### 1.31.1.1.2.32 TPaletteToolList.OnDragDrop Property

Occurs when the user drops an object being dragged.

```
property OnDragDrop;
```

#### Description

Use the OnDragDrop event handler to specify what happens when the user drops an object. The Source parameter of the OnDragDrop event is the object being dropped, and the Sender is the control the object is being dropped on. The X and Y parameters are the coordinates of the mouse positioned over the control.

### 1.31.1.1.2.33 TPaletteToolList.OnDragOver Property

Occurs when the user drags an object over a control.

```
property OnDragOver;
```

#### Description

Use an OnDragOver event to signal that the control can accept a dragged object so the user can drop or dock it.

Within the OnDragOver event handler, change the Accept parameter to false to reject the dragged object. Leave Accept as true to allow the user to drop or dock the dragged object on the control.

To change the shape of the cursor, indicating that the control can accept the dragged object, change the value of the DragCursor property for the control before the OnDragOver event occurs.

The Source is the object being dragged, the Sender is the potential drop or dock site, and X and Y are screen coordinates in pixels. The State parameter specifies how the dragged object is moving over the control.

**Note:** Within the OnDragOver event handler, the Accept parameter defaults to true. However, if an OnDragOver event handler is not supplied, the control rejects the dragged object, as if the Accept parameter were changed to false.

### 1.31.1.1.2.34 TPaletteToolList.OnEndDrag Property

Occurs when the dragging of an object ends, either by dropping the object or by canceling the dragging.

```
property OnEndDrag;
```

#### Description

Use the OnEndDrag event handler to specify any special processing that occurs when dragging stops.

### 1.31.1.1.2.35 TPaletteToolList.OnEnter Property

Occurs when a control receives the input focus.

```
property OnEnter;
```

**Description**

Use the OnEnter event handler to cause any special processing to occur when a control becomes active.

The OnEnter event does not occur when switching between forms or between another application and the application that includes the control.

**1.31.1.1.2.36 TPaletteToolList.OnExit Property**

Occurs when the input focus shifts away from one control to another.

```
property OnExit;
```

**Description**

Use the OnExit event handler to provide special processing when the control ceases to be active.

The OnExit event does not occur when switching between forms or between another application and your application.

**1.31.1.1.2.37 TPaletteToolList.OnKeyDown Property**

Occurs when a user presses any key while the control has focus.

```
property OnKeyDown;
```

**Description**

Use the OnKeyDown event handler to specify special processing to occur when a key is pressed. The OnKeyDown handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys, and pressed mouse buttons.

The TKeyEvent type points to a method that handles keyboard events.

The Key parameter is the key on the keyboard. For non-alphanumeric keys, use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

**1.31.1.1.2.38 TPaletteToolList.OnKeyPress Property**

Occurs when key pressed.

```
property OnKeyPress;
```

**Description**

Use the OnKeyPress event handler to make something happen as a result of a single character key press.

The Key parameter in the OnKeyPress event handler is of type Char; therefore, the OnKeyPress event registers the ASCII character of the key pressed. Keys that don't correspond to an ASCII Char value (Shift or F1, for example) don't generate an OnKeyPress event. Key combinations (such as Shift+A), generate only one OnKeyPress event (for this example, Shift+A results in a Key value of "A" if Caps Lock is off). To respond to non-ASCII keys or key combinations, use the OnKeyDown or OnKeyUp event handlers.

**1.31.1.1.2.39 TPaletteToolList.OnKeyUp Property**

Occurs when the user releases a key that has been pressed.

```
property OnKeyUp;
```

**Description**

Use the OnKeyUp event handler to provide special processing that occurs when a key is released. The OnKeyUp handler can respond to all keyboard keys, including function keys and keys combined with the Shift, Alt, and Ctrl keys.

The TKeyEvent type points to a method that handles keyboard events. The Key parameter is the key on the keyboard. For non-alphanumeric keys, you must use virtual key codes to determine the key pressed. For more information, see Virtual Key codes.

The Shift parameter indicates whether the Shift, Alt, or Ctrl keys are combined with the keystroke.

**1.31.1.1.2.40 TPaletteToolList.OnMouseDown Property**

Occurs when the user presses a mouse button with the mouse pointer over a control.

**property** OnMouseDown;

**Description**

Use the OnMouseDown event handler to implement any special processing that should occur as a result of pressing a mouse button.

The OnMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

**1.31.1.1.2.41 TPaletteToolList.OnMouseMove Property**

Occurs when the user moves the mouse pointer while the mouse pointer is over a control.

**property** OnMouseMove;

**Description**

Use the OnMouseMove event handler to respond when the mouse pointer moves after the control has captured the mouse.

Use the Shift parameter of the OnMouseDown event handler, to determine to the state of the shift keys and mouse buttons. Shift keys are the Shift, Ctrl, and Alt keys or shift key-mouse button combinations. X and Y are pixel coordinates of the new location of the mouse pointer in the client area of the Sender.

**1.31.1.1.2.42 TPaletteToolList.OnMouseUp Property**

Occurs when the user releases a mouse button that was pressed with the mouse pointer over a component.

**property** OnMouseUp;

**Description**

Use an OnMouseUp event handler to implement special processing when the user releases a mouse button.

The OnMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. X and Y are the pixel coordinates of the mouse pointer in the client area of the Sender.

**1.31.1.1.2.43 TPaletteToolList.OnResize Property**

Occurs immediately after the control is resized.

**property** OnResize;

**Description**

Use OnResize to make any final adjustments after a control is resized.

To modify the way a control responds when an attempt is made to resize it, use OnCanResize or OnConstrainedResize.

**1.31.1.1.2.44 TPaletteToolList.OnStartDrag Property**

Occurs when the user begins to drag the control or an object it contains by left-clicking on the control and holding the mouse button down.

```
property OnStartDrag;
```

**Description**

Use the OnStartDrag event handler to implement special processing when the user starts to drag the control or an object it contains. OnStartDrag only occurs if DragKind is dkDrag.

Sender is the control that is about to be dragged, or that contains the object about to be dragged.

The OnStartDrag event handler can create a TDragControlObjectEx instance for the DragObject parameter to specify the drag cursor, or, optionally, a drag image list. If you create a TDragControlObjectEx instance, there is no need to call the Free method for the DragObject when dragging is over. If you create, instead, a TDragControlObject instance, your application is responsible for freeing the drag object instance.

If the OnStartDrag event handler sets the DragObject parameter to nil (Delphi) or NULL (C++), a TDragControlObject object is automatically created and dragging begins on the control itse

**1.31.1.1.2.45 TPaletteToolList.ParentBiDiMode Property**

Specifies whether the control uses its parent's BiDiMode.

```
property ParentBiDiMode;
```

**1.31.1.1.2.46 TPaletteToolList.ParentColor Property**

Determines where a control looks for its color information.

```
property ParentColor;
```

**1.31.1.1.2.47 TPaletteToolList.ParentCtl3D Property**

Determines where a component looks to determine if it should appear three dimensional.

```
property ParentCtl3D;
```

**Description**

ParentCtl3D is provided for backwards compatibility. It has no effect on 32-bit versions of Windows or NT 4.0 and later.

ParentCtl3D determines whether the control uses its parent's Ctl3D property.

**1.31.1.1.2.48 TPaletteToolList.ParentFont Property**

Determines where a control looks for its font information.

```
property ParentFont;
```

#### 1.31.1.1.2.49 TPaletteToolList.ParentShowHint Property

Determines where a control looks to find out if its Help Hint should be shown.

```
property ParentShowHint;
```

#### 1.31.1.1.2.50 TPaletteToolList.PopupMenu Property

Identifies the pop-up menu associated with the control.

```
property PopupMenu;
```

##### Description

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the control and clicks the right mouse button. If the TPopupMenu's AutoPopup property is true, the pop-up menu appears automatically. If the menu's AutoPopup property is false, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

#### 1.31.1.1.2.51 TPaletteToolList.RowSpace Property

Specifies space between two sequential items.

```
property RowSpace: integer;
```

#### 1.31.1.1.2.52 TPaletteToolList.ShowCaptions Property

Specifies whether button captions displayed in tool list.

```
property ShowCaptions: Boolean;
```

#### 1.31.1.1.2.53 TPaletteToolList.ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

```
property ShowHint;
```

#### 1.31.1.1.2.54 TPaletteToolList.StyleCategory Property

Specifies style of category items.

```
property StyleCategory: TToolItemStyle;
```

#### 1.31.1.1.2.55 TPaletteToolList.StyleCategoryMouseOver Property

Specifies style of category item when mouse is over it.

```
property StyleCategoryMouseOver: TToolItemStyle;
```

#### 1.31.1.1.2.56 TPaletteToolList.StyleCategorySelected Property

Specifies style of selected category item.

```
property StyleCategorySelected: TToolItemStyle;
```

#### 1.31.1.1.2.57 TPaletteToolList.StyleItem Property

Specifies style of tool items.

```
property StyleItem: TToolItemStyle;
```

#### 1.31.1.1.2.58 TPaletteToolList.StyleItemMouseOver Property

Specifies style of tool item when mouse is over it.

```
property StyleItemMouseOver: TToolItemStyle;
```

#### 1.31.1.1.2.59 TPaletteToolList.StyleItemSelected Property

Specifies style of selected tool item.

```
property StyleItemSelected: TToolItemStyle;
```

#### 1.31.1.1.2.60 TPaletteToolList.TabOrder Property

Indicates the position of the control in its parent's tab order.

```
property TabOrder;
```

#### 1.31.1.1.2.61 TPaletteToolList.TabStop Property

Determines if the user can tab to a control.

```
property TabStop;
```

##### Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

#### 1.31.1.1.2.62 TPaletteToolList.TransparentImages Property

Specifies whether bottom-left pixel is transparent color. If TransparentImages is False, images are drawn without transparency.

```
property TransparentImages: Boolean;
```

#### 1.31.1.1.2.63 TPaletteToolList.VerticalGroups Property

Specifies whether category item should be displayed vertically along owned items.

```
property VerticalGroups: Boolean;
```

#### 1.31.1.1.2.64 TPaletteToolList.Visible Property

Determines whether the component appears on screen.

```
property Visible;
```

##### Description

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

Calling the Show method sets the control's Visible property to true. Calling the Hide method sets it to false.

### 1.31.1.1.3 TPaletteToolList Events

#### 1.31.1.1.3.1 TPaletteToolList.OnPalChange Event

Occurs when component palette was changed.

```
property OnPalChange: TNotifyEvent;
```

---

## 1.32 eddDsnOpt Namespace

# 1.32.1 Classes

The following table lists classes in this documentation.

## Classes

Class	Description
TDsnOptionsDlg (🔗 see page 490)	Designer (🔗 see page 491) Options dialog box.

## 1.32.1.1 TDsnOptionsDlg Class

Designer (🔗 see page 491) Options dialog box.

### Class Hierarchy



```
TDsnOptionsDlg = class(TForm);
```

### File

eddDsnOpt

### Description

Use the Designer (🔗 see page 491) Options dialog box to specify preferences on the Form Designer (🔗 see page 491).

<b>Display grid</b>	Displays dots on the form to make the grid visible
<b>Snap to grid</b>	Automatically aligns components on the form with the nearest gridline. You cannot place a component "in between" gridlines
<b>Grid size X</b>	Sets grid spacing in pixels along the x-axis. Specify a higher number (between 2 and 128) to increase grid spacing
<b>Grid size Y</b>	Sets grid spacing in pixels along the y-axis. Specify a higher number (between 2 and 128) to increase grid spacing
<b>Multiple selection</b>	Allows to select more than one component at a time
<b>Show component captions</b>	Displays captions for nonvisual components you drop on a form or data module
<b>Flat component icons</b>	Flattens icons for nonvisual components
<b>Show invisible components</b>	Display nonvisual components
<b>Caption font</b>	Specifies attributes for component caption font
<b>Grab color</b>	Specifies color for the grab markers
<b>Grab size</b>	Specifies size of the grab markers
<b>Show designer hints</b>	Displays a class name in a Help tooltip for a nonvisual component you drop on form or data module, origin and size. Note that this option only affects tooltips that appear when you pause the mouse over a component. Help tooltips are always enabled in the Component palette

### Members



#### TDsnOptionsDlg Properties

TDsnOptionsDlg Properties	Description
Designer (🔗 see page 491)	Specifies designer which properties are edited.

Legend

	Property
-----------------------------------------------------------------------------------	----------

TDsnOptionsDlg Properties

TDsnOptionsDlg Properties	Description
 Designer (  see page 491)	Specifies designer which properties are edited.

1.32.1.1.1 TDsnOptionsDlg Properties

1.32.1.1.1.1 TDsnOptionsDlg.Designer Property

Specifies designer which properties are edited.


```
property Designer: TzFormDesigner;
```

1.33 eddObjInspProp Namespace

1.33.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TObjInspPropDlg (  see page 491)	This dialog box displays properties for Object Inspector

1.33.1.1 TObjInspPropDlg Class

This dialog box displays properties for Object Inspector

Class Hierarchy



```
TObjInspPropDlg = class(TForm);
```

File

eddObjInspProp

Description

Use this page to specify options for the Object Inspector, which you can also access by right-clicking the Object Inspector and choosing Properties.

SpeedSettings

Click the drop-down list box to import, choose, and customize settings from the following color schemes: Custom, Default, Delphi 5, and Visual Studio.



## Colors

To customize one of the imported color schemes, select it from the SpeedSettings list. Then select an option and select a different color from the drop-down list below. For example, to change the color of Value, the text color for properties' values, select Value and click cYellow from the Options list. You save your new settings once you click OK. This automatically saves the changes to the Custom colors and settings scheme, not the original imported one.

To return to your default settings, click Default colors and settings or one of the others.

## Options

Sets preferences for displaying several options on the Object Inspector.

Check box	When checked
<b>Show instance class</b>	Displays the drop-down list box of components and their class names (called the instance list) at the top of the Object Inspector. The list is useful when you have many components on your form or data module and can't find the one you want right away. Click the drop-down arrow and select the component you want to focus on. To hide the list, uncheck this check box.
<b>Show classname in instance list</b>	Displays the component's class name for every component in the instance list, not just the first one
<b>Show status bar</b>	Displays the status bar at the bottom of the Object Inspector. The status bar states how many properties or events are not shown as a result of using the View command. If all properties or events are visible in the Object Inspector, it says "All shown."
<b>Render background grid</b>	Adds horizontal background lines to designate columns and rows on the Properties and Events pages.
<b>Integral height (when not docked)</b>	As you vertically resize the Object Inspector with your cursor, this option adjusts the Object Inspector between a full row instead of a partial row.
<b>Show read only properties</b>	Displays the properties for components (usually third-party) even if the properties are read only. By default, they are grayed out.



## References

Component references are properties that are also components. Once you add the referenced component to your form and refer the first component to it, you can view and edit the referenced component's properties without selecting it on the form. For example, if you add an ActionList and ImageList component to your form and set the ActionList's Images property to ImageList1, the Object Inspector displays the ImageList's properties.

Check box	When checked
<b>Expand inline</b>	Displays the properties of the referenced component. To view these properties, click the plus (+) sign next to the referenced component. By default, referenced components are red and their properties green.
<b>Show events on page</b>	Displays the events of the referenced component. By default, referenced properties are red and their events green.

Members



TObjInspPropDlg Properties

TObjInspPropDlg Properties	Description
 ObjectInspector (  see page 493)	This is ObjectInspector, a member of class TObjInspPropDlg.

Legend

	Property
-----------------------------------------------------------------------------------	----------

TObjInspPropDlg Properties

TObjInspPropDlg Properties	Description
 ObjectInspector (  see page 493)	This is ObjectInspector, a member of class TObjInspPropDlg.

1.33.1.1.1 TObjInspPropDlg Properties

1.33.1.1.1.1 TObjInspPropDlg.ObjectInspector Property

```
property ObjectInspector: TObjectInspectorFrame;
```

Description


This is ObjectInspector, a member of class TObjInspPropDlg.

1.34 eddPackageCtrl Namespace

1.34.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TPackageCtrlDlg (  see page 493)	Use this dialog box to specify the design-time packages installed in the design-mode and the runtime packages required by them.

1.34.1.1 TPackageCtrlDlg Class

Use this dialog box to specify the design-time packages installed in the design-mode and the runtime packages required by them.

Class Hierarchy



```
TPackageCtrlDlg = class(TForm);
```

File

eddPackageCtrl

Description

This dialog box is absolutely similar to standard Borland' Packages page.

[See it for details](#)

# 1.35 eddPageName Namespace

## 1.35.1 Classes

The following table lists classes in this documentation.

**Classes**

Class	Description
TPageNameDlg (🔗 see page 494)	Use this dialog box to specify a new name for a page on the Component palette.

### 1.35.1.1 TPageNameDlg Class

Use this dialog box to specify a new name for a page on the Component palette.

**Class Hierarchy**



```
TPageNameDlg = class(TForm);
```

**File**

eddPageName

**Description**

This dialog box is absolutely similar to standard Borland' Page Name dialog.

[See it for details](#)

# 1.36 eddObjInspFrm Namespace

## 1.36.1 Classes

The following table lists classes in this documentation.

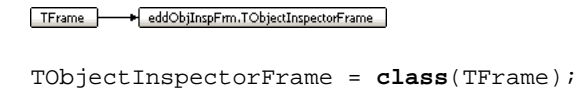
**Classes**

Class	Description
TObjectInspectorFrame (🔗 see page 495)	The Object Inspector is the connection between your application's visual appearance and the code that makes your application run.

### 1.36.1.1 TObjectInspectorFrame Class

The Object Inspector is the connection between your application's visual appearance and the code that makes your application run.

**Class Hierarchy**



**File**

eddObjInspFrm

**Description**

The Object Inspector enables you to:

- Set design-time properties for components you have placed on a form (or for the form itself).
- Filter visible properties and events.

The object selector, or instance list, at the top of the Object Inspector is a drop-down list containing all the components on the active form and it also displays the object type, or class, of the selected component. This lets you quickly display properties and events for the different components on the current form.

You can resize the columns of the Object Inspector by dragging the separator line to a new position.



The Object Inspector has two pages:

- Properties page
  - Events page
- Object Inspector tabs provide a means to switch between the Property page and the Events page of the Object Inspector. To change pages, click the Properties or Events tab.




You can display and filter properties and events by category. By filtering the properties, you can reduce the number of properties visible in the Object Inspector and focus on those which are primarily of interest at the time. You can also more easily locate related properties by viewing them by category. For example, when localizing your application for other countries, you can display only properties that need to be localized by unchecking all categories except Localizable. See Property and event categories in the Object Inspector.













**Members**

**TObjectInspectorFrame Methods**





TObjectInspectorFrame Methods	Description
 Create (see page 496)	Creates and initializes a TObjectInspectorFrame instance.
 Customize (see page 497)	Displays "Object Inspector Properties" dialog to change object inspector appearance.

**TObjectInspectorFrame Properties**






TObjectInspectorFrame Properties	Description
 ComponentCombo (see page 497)	References nested component combo-box.
 EventsList (see page 497)	References nested inspector list with events.
 IntegralHeight (see page 497)	Specifies whether height of object inspector frame should be adjusted to inspector lists show full row instead of a partial row.

 <b>PageControl</b> (  see page 497)	References nested page control.
 <b>Pages</b> (  see page 497)	Specifies visible tab sheets in object inspector. "Properties" tab contains inspector list with properties; "Events" tab contains inspector list with procedural properties, i.e. with events.
 <b>PropertyList</b> (  see page 497)	References nested inspector list with properties.
 <b>ReadOnly</b> (  see page 497)	Specifies whether inspector lists are in read-only state.
 <b>ShowInstanceList</b> (  see page 497)	Specifies whether component combo-box of object inspector is visible.
 <b>ShowStatusBar</b> (  see page 497)	Specifies whether status bar of object inspector is visible.





## TObjectInspectorFrame Events

TObjectInspectorFrame Events	Description
 <b>OnHideClick</b> (  see page 498)	Occurs when user clicks "Hide" in popup menu.
 <b>OnStayOnTopClick</b> (  see page 498)	Occurs when user click "Stay on top" in popup menu.






## Legend

	Constructor
	virtual
	Property
	read only
	Event



















## TObjectInspectorFrame Events

TObjectInspectorFrame Events	Description
 <b>OnHideClick</b> (  see page 498)	Occurs when user clicks "Hide" in popup menu.
 <b>OnStayOnTopClick</b> (  see page 498)	Occurs when user click "Stay on top" in popup menu.

## TObjectInspectorFrame Methods

TObjectInspectorFrame Methods	Description
  <b>Create</b> (  see page 496)	Creates and initializes a TObjectInspectorFrame instance.
 <b>Customize</b> (  see page 497)	Displays "Object Inspector Properties" dialog to change object inspector appearance.

## TObjectInspectorFrame Properties

TObjectInspectorFrame Properties	Description
 <b>ComponentCombo</b> (  see page 497)	References nested component combo-box.
 <b>EventsList</b> (  see page 497)	References nested inspector list with events.
 <b>IntegralHeight</b> (  see page 497)	Specifies whether height of object inspector frame should be adjusted to inspector lists show full row instead of a partial row.
 <b>PageControl</b> (  see page 497)	References nested page control.
 <b>Pages</b> (  see page 497)	Specifies visible tab sheets in object inspector. "Properties" tab contains inspector list with properties; "Events" tab contains inspector list with procedural properties, i.e. with events.
 <b>PropertyList</b> (  see page 497)	References nested inspector list with properties.
 <b>ReadOnly</b> (  see page 497)	Specifies whether inspector lists are in read-only state.
 <b>ShowInstanceList</b> (  see page 497)	Specifies whether component combo-box of object inspector is visible.
 <b>ShowStatusBar</b> (  see page 497)	Specifies whether status bar of object inspector is visible.

## 1.36.1.1.1 TObjectInspectorFrame Methods

### 1.36.1.1.1.1 TObjectInspectorFrame.Create Constructor

Creates and initializes a TObjectInspectorFrame instance.

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

#### Description

Use Create to programmatically instantiate a TObjectInspectorFrame component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

### 1.36.1.1.1.2 TObjectInspectorFrame.Customize Method

Displays "Object Inspector Properties" dialog to change object inspector appearance.

```
function Customize: Boolean;
```

### 1.36.1.1.2 TObjectInspectorFrame Properties

#### 1.36.1.1.2.1 TObjectInspectorFrame.ComponentCombo Property

References nested component combo-box.

```
property ComponentCombo: TComponentCombo;
```

#### 1.36.1.1.2.2 TObjectInspectorFrame.EventsList Property

References nested inspector list with events.

```
property EventsList: TInspectorList;
```

#### 1.36.1.1.2.3 TObjectInspectorFrame.IntegralHeight Property

Specifies whether height of object inspector frame should be adjusted to inspector lists show full row instead of a partial row.

```
property IntegralHeight: Boolean;
```

#### 1.36.1.1.2.4 TObjectInspectorFrame.PageControl Property

References nested page control.

```
property PageControl: TPageControl;
```

#### 1.36.1.1.2.5 TObjectInspectorFrame.Pages Property

Specifies visible tab sheets in object inspector. "Properties" tab contains inspector list with properties; "Events" tab contains inspector list with procedural properties, i.e. with events.

```
property Pages: TObjInspTabs;
```

#### 1.36.1.1.2.6 TObjectInspectorFrame.PropertyList Property

References nested inspector list with properties.

```
property PropertyList: TInspectorList;
```

#### 1.36.1.1.2.7 TObjectInspectorFrame.ReadOnly Property

Specifies whether inspector lists are in read-only state.

```
property ReadOnly: Boolean;
```

#### 1.36.1.1.2.8 TObjectInspectorFrame.ShowInstanceList Property

Specifies whether component combo-box of object inspector is visible.

```
property ShowInstanceList: Boolean;
```

#### 1.36.1.1.2.9 TObjectInspectorFrame.ShowStatusBar Property

Specifies whether status bar of object inspector is visible.

```
property ShowStatusBar: Boolean;
```

### 1.36.1.1.3 TObjectInspectorFrame Events

### 1.36.1.1.3.1 TObjectInspectorFrame.OnHideClick Event

Occurs when user clicks "Hide" in popup menu.

```
property OnHideClick: TNotifyEvent;
```

### 1.36.1.1.3.2 TObjectInspectorFrame.OnStayOnTopClick Event

Occurs when user click "Stay on top" in popup menu.

```
property OnStayOnTopClick: TNotifyEvent;
```

## 1.36.2 Types

The following table lists types in this documentation.

### Types

Type	Description
TObjInspTabs (🔗 see page 498)	Specifies visible tab sheets in object inspector. "Properties" tab contains inspector list with properties; "Events" tab contains inspector list with procedural properties, i.e. with events.

### 1.36.2.1 eddObjInspFrm.TObjInspTabs Type

Specifies visible tab sheets in object inspector. "Properties" tab contains inspector list with properties; "Events" tab contains inspector list with procedural properties, i.e. with events.

```
TObjInspTabs = set of (oi_Properties, oi_Events);
```

### File

eddObjInspFrm

## 1.37 eddScaleDI Namespace

### 1.37.1 Classes

The following table lists classes in this documentation.

### Classes

Class	Description
TScaleDlg (🔗 see page 498)	Scale dialog box

### 1.37.1.1 TScaleDlg Class

Scale dialog box

### Class Hierarchy



```
TScaleDlg = class(TForm);
```

File

eddScaleDI

Description

Use this dialog box to proportionally resize the form and all of its components.

Scaling Factor, In Percent

Enter a percentage to which you want to resize the form. The scaling factor must be between 25 and 400.

Percentages over 100 grow the form.

Percentages under 100 shrink the form.

# 1.38 eddSelFrame Namespace

## 1.38.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TSelFrameDlg (🔗 see page 499)	The Select Frame dialog lists all the frames included in the current project. Choose the frame you want to embed in the form or frame you just clicked on, then press OK.

### 1.38.1.1 TSelFrameDlg Class

The Select Frame dialog lists all the frames included in the current project. Choose the frame you want to embed in the form or frame you just clicked on, then press OK.

Class Hierarchy



```
TSelFrameDlg = class(TForm);
```

File

eddSelFrame

# 1.39 eddObjTreeFrame Namespace



# 1.39.1 Classes

The following table lists classes in this documentation.

## Classes

Class	Description
TObjectTreeFrame (🔗 see page 500)	Represents object tree view with integrated tool bar and standard object tree actions.

## 1.39.1.1 TObjectTreeFrame Class

Represents object tree view with integrated tool bar and standard object tree actions.

### Class Hierarchy



```
TObjectTreeFrame = class(TFrame);
```

### File

eddObjTreeFrame

### Members

#### TObjectTreeFrame Methods

TObjectTreeFrame Methods	Description
🔗🔗🔗 Create (🔗 see page 500)	Creates and initializes a TObjectTreeFrame instance.

#### TObjectTreeFrame Properties

TObjectTreeFrame Properties	Description
🔗🔗🔗 ObjectTree (🔗 see page 501)	Provides access to object tree control.

### Legend

🔗🔗🔗	Constructor
🔗	virtual
🔗🔗	Property
🔗	read only

#### TObjectTreeFrame Methods

TObjectTreeFrame Methods	Description
🔗🔗🔗 Create (🔗 see page 500)	Creates and initializes a TObjectTreeFrame instance.

#### TObjectTreeFrame Properties

TObjectTreeFrame Properties	Description
🔗🔗🔗 ObjectTree (🔗 see page 501)	Provides access to object tree control.

## 1.39.1.1.1 TObjectTreeFrame Methods

### 1.39.1.1.1.1 TObjectTreeFrame.Create Constructor

Creates and initializes a TObjectTreeFrame instance.

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

### Description

Use Create to programmatically instantiate a TObjectTreeFrame component. Components added in the form designer are created automatically.

AOwner is the component that is responsible for freeing the component instance. It becomes the value of the Owner property.

### 1.39.1.1.2 TObjectTreeFrame Properties

#### 1.39.1.1.2.1 TObjectTreeFrame.ObjectTree Property

Provides access to object tree control.

**property** ObjectTree: TDesignerObjTree;

## 1.40 eddSizeDlg Namespace

### 1.40.1 Classes

The following table lists classes in this documentation.

**Classes**

Class	Description
TSizeAdjDlg (🔗 see page 501)	Size dialog box.

#### 1.40.1.1 TSizeAdjDlg Class

Size dialog box.

**Class Hierarchy**



TSizeAdjDlg = **class**(TForm);

**File**

eddSizeDlg

**Description**

Use this dialog box to resize multiple components to be exactly the same height or width.

- The Width options change the horizontal size of the selected components.
- The Height options align the vertical size of the selected components.

The options for horizontal or vertical sizing are:

Option	Description
No change	Does not change the size of the components.
Shrink to smallest	Resizes the group of components to the height or width of the smallest selected component.
Grow to largest	Resizes the group of components to the height or width of the largest selected component.
Width	Sets a custom width for the selected components.
Height	Sets a custom height for the selected component

# 1.41 eddTabOrdDI Namespace

## 1.41.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TTabOrderDlg (📄 see page 502)	Edit Tab Order dialog box

### 1.41.1.1 TTabOrderDlg Class

Edit Tab Order dialog box

Class Hierarchy



```
TTabOrderDlg = class(TForm);
```

File

eddTabOrdDI

Description

Use this dialog box to modify the tab order of the components on the form or within the selected component if that component contains other components.

**Controls** Lists the components on the active form in their current tab order. The first component listed is the first component in the tab order. The default tab order is determined by the order in which you placed the components on the form

To change the tabs order of a component:

1. Select the component name.
2. Click the up button to move the component up in the tab order, or click the down arrow to move it down in the tab order.

You can also drag the selected component to its new position in the tab order.











3. To save your changes, click OK.

# 1.42 edlOUtils Namespace

## 1.42.1 Functions

The following table lists functions in this documentation.

### Functions

Function	Description
 <b>zCopyCmpResource</b> (  see page 504)	Copies properties of Source component to Dest component via memory stream.
 <b>zReadCmpFromFile</b> (  see page 504)	<p>Reads component's resource from the file (forms, data modules, ...). As against of ReadComponentResFile, it performs:</p> <ol style="list-style-type: none"> <li>1. Automatic detection file format (text / binary).</li> <li>2. Existing element is not deleted, it is only updated from the resource.</li> <li>3. Not existing components are created and loaded from resource.</li> <li>4. If loaded component is a registered frame, it is initialized during creation.</li> <li>5. Allows ignoring and processing errors using OnError procedure.</li> <li>6. Allows controlling component creation using OnCreateCmp procedure.</li> </ol> <p>Events - list of event associations, where names are event path, values - script procedures. If Events is <b>nil</b> no event associations are read from form.</p>
 <b>zReadCmpFromStream</b> (  see page 504)	<p>Reads component's resource from the stream (forms, data modules, ...). As against of ReadComponentResFile, it performs:</p> <ol style="list-style-type: none"> <li>1. Automatic detection file format (text / binary).</li> <li>2. Existing element is not deleted, it is only updated from the resource.</li> <li>3. Not existing components are created and loaded from resource.</li> <li>4. If loaded component is a registered frame, it is initialized during creation.</li> <li>5. Allows ignoring and processing errors using OnError procedure.</li> <li>6. Allows controlling component creation using OnCreateCmp procedure.</li> </ol> <p>Events - list of event associations, where names are event path, values - script procedures. If Events is <b>nil</b> no event associations are read from form.</p>
 <b>zWriteCmpToFile</b> (  see page 505)	<p>Write Root component resource to file FileName. Set AsText to True to write in text resource format, otherwise component will be written in binary format.</p> <p>Events - list of event associations, where names are event path, values - script procedures. If events is <b>nil</b> no event associations is saved to form.</p>
 <b>zWriteCmpToStream</b> (  see page 505)	<p>Saves component's resource (forms, data modules, ...) to stream. If AsText is True, saves resource in text format, otherwise saves in binary.</p> <p>Events - list of event associations, where names are event path, values - script procedures. If events is <b>nil</b> no event associations is saved to form.</p>

### Legend

	Method
-------------------------------------------------------------------------------------	--------

### 1.42.1.1 edIUtils.zCopyCmpResource Function

Copies properties of Source component to Dest component via memory stream.

```
procedure zCopyCmpResource(Dest: TComponent; Source: TComponent);
```

#### File

edIUtils

### 1.42.1.2 edIUtils.zReadCmpFromFile Function

Reads component's resource from the file (forms, data modules, ...). As against of ReadComponentResFile, it performs:

1. Automatic detection file format (text / binary).
2. Existing element is not deleted, it is only updated from the resource.
3. Not existing components are created and loaded from resource.
4. If loaded component is a registered frame, it is initialized during creation.
5. Allows ignoring and processing errors using OnError procedure.
6. Allows controlling component creation using OnCreateCmp procedure.

Events - list of event associations, where names are event path, values - script procedures. If Events is **nil** no event associations are read from form.

```
procedure zReadCmpFromFile(const FileName: string; Root: TComponent; OnError: TReaderError = nil; OnCreateCmp: TCreateComponentEvent = nil; Events: TStrings = nil);
```

#### File

edIUtils

### 1.42.1.3 edIUtils.zReadCmpFromStream Function

Reads component's resource from the stream (forms, data modules, ...). As against of ReadComponentResFile, it performs:

1. Automatic detection file format (text / binary).
2. Existing element is not deleted, it is only updated from the resource.
3. Not existing components are created and loaded from resource.
4. If loaded component is a registered frame, it is initialized during creation.
5. Allows ignoring and processing errors using OnError procedure.
6. Allows controlling component creation using OnCreateCmp procedure.

Events - list of event associations, where names are event path, values - script procedures. If Events is **nil** no event associations are read from form.

```
procedure zReadCmpFromStream(Stream: TStream; Root: TComponent; OnError: TReaderError = nil; OnCreateCmp: TCreateComponentEvent = nil; Events: TStrings = nil);
```

#### File

edIUtils

### 1.42.1.4 edIOUtils.zWriteCmpToFile Function

Write Root component resource to file FileName. Set AsText to True to write in text resource format, otherwise component will be written in binary format.

Events - list of event associations, where names are event path, values - script procedures. If events is **nil** no event associations is saved to form.

```
procedure zWriteCmpToFile(const FileName: string; Root: TComponent; AsText: Boolean = True;
Events: TStrings = nil);
```

**File**  
edIOUtils

### 1.42.1.5 edIOUtils.zWriteCmpToStream Function

Saves component's resource (forms, data modules, ...) to stream. If AsText is True, saves resource in text format, otherwise saves in binary.

Events - list of event associations, where names are event path, values - script procedures. If events is **nil** no event associations is saved to form.

```
procedure zWriteCmpToStream(Stream: TStream; Root: TComponent; AsText: Boolean = True;
Events: TStrings = nil);
```

**File**  
edIOUtils

## 1.43 edManager Namespace

### 1.43.1 Classes

The following table lists classes in this documentation.

**Classes**

Class	Description
TDesignerManager <a href="#">[?] see page 505</a>	TDesignerManager is a dispatcher of the design environment events.

#### 1.43.1.1 TDesignerManager Class

TDesignerManager is a dispatcher of the design environment events.

**Class Hierarchy**

edManager.TDesignerManager

```
TDesignerManager = class;
```

**File**  
edManager

## Description

To receive events clients should implement following interfaces: [IClassSelector](#) (see page 511), [IDesignIDEEvents](#) (see page 512), [IDesignNotificatio](#), and add itself to client list using [AddClient](#) (see page 507) method.

Design manager is been created when application starts and stored in global variable [DsnManager](#) (see page 514).






















Use [DsnManager](#) (see page 514) to access to.

Do not create (see page 507) other objects of this class.




Do not destroy (see page 508) it manually.

## Members





### TDesignerManager Methods

TDesignerManager Methods	Description
 <a href="#">AddClient</a> (see page 507)	<a href="#">AddClient</a> adds client to handle <a href="#">TDesignerManager</a> events.
 <a href="#">BeforeRegisterComponent</a> (see page 507)	Called before registering component class.
 <a href="#">Create</a> (see page 507)	Creates and initializes instance of the <a href="#">TDesignerManager</a> class
 <a href="#">CreateCurrent</a> (see page 507)	Creates instance in active designer of currently selected component class and places this component at the center of form.
 <a href="#">DesignerClosed</a> (see page 507)	Called when designer <a href="#">ADesigner</a> has been deactivated.
 <a href="#">DesignerOpened</a> (see page 508)	Called when designer <a href="#">ADesigner</a> has been activated.
  <a href="#">Destroy</a> (see page 508)	Destroys <a href="#">DsnManager</a> (see page 514) object and frees memory.
 <a href="#">GetGlobalComponents</a> (see page 508)	Called to request global components.
 <a href="#">GetWorkspaceOrigin</a> (see page 508)	Called to request workspace origin.
 <a href="#">ItemDeleted</a> (see page 508)	Called after object has been deleted in the active designer.
 <a href="#">ItemInserted</a> (see page 508)	Called after object has been inserted in the active designer.
 <a href="#">ItemsModified</a> (see page 509)	Called after selected objects have been modified in the active designer
 <a href="#">KeyDown</a> (see page 509)	Called when <a href="#">KeyDown</a> event occurs in active designer.
 <a href="#">KeyPress</a> (see page 509)	Called when <a href="#">KeyPress</a> event occurs in active designer.
 <a href="#">PaletteChanged</a> (see page 509)	Called when contents of th component palette is changed.
 <a href="#">RemoveClient</a> (see page 509)	Removes specified client from the list
 <a href="#">ResetCmpClass</a> (see page 509)	Called after inserting component in the form. If <a href="#">MultiCreate</a> (see page 511) = <a href="#">False</a> , <a href="#">ComponentClass</a> (see page 510) will be reset to nil.
 <a href="#">SelectionChanged</a> (see page 509)	Called after selection have been changed in the active designer.
  <a href="#">SetActiveDesigner</a> (see page 510)	Sets active designer.







### TDesignerManager Properties

TDesignerManager Properties	Description
 <a href="#">ActiveDesigner</a> (see page 510)	Current active designer.
 <a href="#">ComponentClass</a> (see page 510)	Current component class selected in the component palette.
 <a href="#">MultiCreate</a> (see page 511)	Specifies whether <a href="#">ComponentClass</a> (see page 510) must be reset to nil after component has been added to the form in the designer.

## Legend

	Method
	virtual
	protected
	Property

### TDesignerManager Methods

TDesignerManager Methods	Description
 <a href="#">AddClient</a> (see page 507)	<a href="#">AddClient</a> adds client to handle <a href="#">TDesignerManager</a> events.
 <a href="#">BeforeRegisterComponent</a> (see page 507)	Called before registering component class.
 <a href="#">Create</a> (see page 507)	Creates and initializes instance of the <a href="#">TDesignerManager</a> class
 <a href="#">CreateCurrent</a> (see page 507)	Creates instance in active designer of currently selected component class and places this component at the center of form.
 <a href="#">DesignerClosed</a> (see page 507)	Called when designer <a href="#">ADesigner</a> has been deactivated.
 <a href="#">DesignerOpened</a> (see page 508)	Called when designer <a href="#">ADesigner</a> has been activated.

✦ Destroy (see page 508)	Destroys DsnManager (see page 514) object and frees memory.
✦ GetGlobalComponents (see page 508)	Called to request global components.
✦ GetWorkspaceOrigin (see page 508)	Called to request workspace origin.
✦ ItemDeleted (see page 508)	Called after object has been deleted in the active designer.
✦ ItemInserted (see page 508)	Called after object has been inserted in the active designer.
✦ ItemsModified (see page 509)	Called after selected objects have been modified in the active designer
✦ KeyDown (see page 509)	Called when KeyDown event occurs in active designer.
✦ KeyPress (see page 509)	Called when KeyPress event occurs in active designer.
✦ PaletteChanged (see page 509)	Called when contents of th component palette is changed.
✦ RemoveClient (see page 509)	Removes specified client from the list
✦ ResetCmpClass (see page 509)	Called after inserting component in the form. If MultiCreate (see page 511) = False, ComponentClass (see page 510) will be reset to nil.
✦ SelectionChanged (see page 509)	Called after selection have been changed in the active designer.
✦ SetActiveDesigner (see page 510)	Sets active designer.

## TDesignerManager Properties

TDesignerManager Properties	Description
ActiveDesigner (see page 510)	Current active designer.
ComponentClass (see page 510)	Current component class selected in the component palette.
MultiCreate (see page 511)	Specifies whether ComponentClass (see page 510) must be reset to nil after component has been added to the form in the designer.

## 1.43.1.1.1 TDesignerManager Methods

### 1.43.1.1.1.1 TDesignerManager.AddClient Method

AddClient adds client to handle TDesignerManager (see page 505) events.

```
procedure AddClient(const Client: IUnknown);
```

#### Description

Client may implement any designer's environment interfaces (IClassSelector (see page 511), IDesignIDEEvents (see page 512), IDesignNotification), through which client will receive designer's events.

### 1.43.1.1.1.2 TDesignerManager.BeforeRegisterComponent Method

Called before registering component class.

```
procedure BeforeRegisterComponent(CompClass: TComponentClass; var Page: string; var Accept: Boolean);
```

### 1.43.1.1.1.3 TDesignerManager.Create Constructor

Creates and initializes instance of the TDesignerManager (see page 505) class

```
constructor Create;
```

#### Description

Do not create objects of this class directly.

Use global variable DsnManager (see page 514) instead.

### 1.43.1.1.1.4 TDesignerManager.CreateCurrent Method

Creates instance in active designer of currently selected component class and places this component at the center of form.

```
procedure CreateCurrent;
```

### 1.43.1.1.1.5 TDesignerManager.DesignerClosed Method

Called when designer ADesigner has been deactivated.



```
procedure DesignerClosed(const ADesigner: IDesigner; AGoingDormant: Boolean);
```

**Description**

Client must implement IDesignNotification interface to catch this event.

**1.43.1.1.1.6 TDesignerManager.DesignerOpened Method**

Called when designer ADesigner has been activated.

```
procedure DesignerOpened(const ADesigner: IDesigner; AResurrecting: Boolean);
```

**Description**

Client must implement IDesignNotification interface to catch this event.

**1.43.1.1.1.7 TDesignerManager.Destroy Destructor**

Destroys DsnManager (see page 514) object and frees memory.

```
destructor Destroy; override;
```

**Description**

Do not destroy it directly.

DsnManager (see page 514) destroys automatically when application finishes.

**1.43.1.1.1.8 TDesignerManager.GetGlobalComponents Method**

Called to request global components.

```
procedure GetGlobalComponents(Root: TComponent; List: TList);
```

**Description**

Client must implement IDesignIDEEvents (see page 512) interface to catch this event.

**1.43.1.1.1.9 TDesignerManager.GetWorkspaceOrigin Method**

Called to request workspace origin.

```
function GetWorkspaceOrigin: TPoint;
```

**Description**

Client must implement IDesignIDEEvents (see page 512) interface to catch this event.

**1.43.1.1.1.10 TDesignerManager.ItemDeleted Method**

Called after object has been deleted in the active designer.

```
procedure ItemDeleted(const ADesigner: IDesigner; AItem: TPersistent);
```

**Description**

Client must implement IDesignNotification interface to catch this event

**1.43.1.1.1.11 TDesignerManager.ItemInserted Method**

Called after object has been inserted in the active designer.

```
procedure ItemInserted(const ADesigner: IDesigner; AItem: TPersistent);
```

**Description**

Client must implement IDesignNotification interface to catch this event

#### 1.43.1.1.1.12 TDesignerManager.ItemsModified Method

Called after selected objects have been modified in the active designer

```
procedure ItemsModified(const ADesigner: IDesigner);
```

##### Description

Client must implement IDesignNotification interface to catch this event

#### 1.43.1.1.1.13 TDesignerManager.KeyDown Method

Called when KeyDown event occurs in active designer.

```
procedure KeyDown(Sender: IDesigner; var Key: Word; Shift: TShiftState);
```

##### Description

Client must implement IDesignIDEEvents (see page 512) interface to catch this event

#### 1.43.1.1.1.14 TDesignerManager.KeyPress Method

Called when KeyPress event occurs in active designer.

```
procedure KeyPress(Sender: IDesigner; var Key: Char);
```

##### Description

Client must implement IDesignIDEEvents (see page 512) interface to catch this event

#### 1.43.1.1.1.15 TDesignerManager.PaletteChanged Method

Called when contents of the component palette is changed.

```
procedure PaletteChanged;
```

##### Description

Client must implement IClassSelector (see page 511) interface to catch this event

#### 1.43.1.1.1.16 TDesignerManager.RemoveClient Method

Removes specified client from the list

```
procedure RemoveClient(const Client: IUnknown);
```

##### Description

#### 1.43.1.1.1.17 TDesignerManager.ResetCmpClass Method

Called after inserting component in the form. If MultiCreate (see page 511) = False, ComponentClass (see page 510) will be reset to nil.

```
procedure ResetCmpClass;
```

##### Description

#### 1.43.1.1.1.18 TDesignerManager.SelectionChanged Method

Called after selection have been changed in the active designer.

```
procedure SelectionChanged(const ADesigner: IDesigner; const ASelection: IDesignerSelections);
```

**Description**

Client must implement IDesignNotification interface to catch this event

**1.43.1.1.1.19 TDesignerManager.SetActiveDesigner Method**

Sets active designer.

```
procedure SetActiveDesigner(const Value: IDesigner); virtual;
```

**Description**

Override this method when you need to switch managers.

Application:

There are several groups of designers (edited objects) with own component palette state and design objects.

Usage:

For each group there should be created own instance of TDesignManager (derived class).

Derive new class from TDesignManager and override SetActiveDesigner.

In SetActiveDesigner you should detect in which group this active designer is included and change DsnManager (see page 514) global variable.

```
procedure TMyDesignManager.SetActiveDesigner(const Value: IDesigner);
var Mng: TMyDesignManager;
begin
    // GetManagerForDesigner - your function which detects particular group
    Mng := GetManagerForDesigner(Value);
    if Mng <> Self then
    begin
        DsnManager := Mng;
        Mng.SetActiveDesigner(Value);
    end else
        inherited;
end;
```

**1.43.1.1.2 TDesignerManager Properties****1.43.1.1.2.1 TDesignerManager.ActiveDesigner Property**

Current active designer.

```
property ActiveDesigner: IDesigner;
```

**Description**

Read this property to get reference to the currently active designer.

Set this property to change it.

**1.43.1.1.2.2 TDesignerManager.ComponentClass Property**

Current component class selected in the component palette.

```
property ComponentClass: TComponentClass;
```

**Description**

Use this property to get what type of class is being selected in the component palette.

Set this property to change this TComponentClass.

This TComponentClass type is used when creating new components on the designed form.

### 1.43.1.1.2.3 TDesignerManager.MultiCreate Property

Specifies whether ComponentClass (see page 510) must be reset to nil after component has been added to the form in the designer.

```
property MultiCreate: Boolean;
```

Description

If MultiCreate = True, user may create (see page 507) several instances of selected class without selection in the palette. This mode is activated when user selects component on the palette with pressed Shift.

## 1.43.2 Interfaces

The following table lists interfaces in this documentation.

Interfaces

Interface	Description
↔ IClassSelector (see page 511)	Interface to accept component palette events.
↔ IDesignIDEEvents (see page 512)	Interface to accept designer events.

Legend

↔	Interface
---	-----------

### 1.43.2.1 IClassSelector Interface

Interface to accept component palette events.

Class Hierarchy

```
edManager.IClassSelector
```

```
IClassSelector = interface;
```

File

edManager

Members

IClassSelector Methods

IClassSelector Methods	Description
↔ ClsChanged (see page 511)	Called when selected in component palette component class was changed.
↔ ClsPalChanged (see page 512)	Called when component palette was changed.

Legend

↔	Method
---	--------

IClassSelector Methods

IClassSelector Methods	Description
↔ ClsChanged (see page 511)	Called when selected in component palette component class was changed.
↔ ClsPalChanged (see page 512)	Called when component palette was changed.

#### 1.43.2.1.1 IClassSelector Methods

##### 1.43.2.1.1.1 IClassSelector.ClsChanged Method

Called when selected in component palette component class was changed.

```
procedure ClsChanged;
```

### 1.43.2.1.1.2 IClassSelector.ClsPalChanged Method

Called when component palette was changed.

```
procedure ClsPalChanged;
```

## 1.43.2.2 IDesignIDEEvents Interface

Interface to accept designer events.

### Class Hierarchy

```
edManager.IDesignIDEEvents
```

```
IDesignIDEEvents = interface;
```

### File

edManager

### Members

#### IDesignIDEEvents Methods

IDesignIDEEvents Methods	Description
◆ ActiveDsnChanged (see page 512)	Called when active designer was changed.
◆ BeforeRegisterComponent (see page 512)	Called before registering component class.
◆ GetGlobalComponents (see page 512)	Called to get global components.
◆ GetWorkspaceOrigin (see page 513)	Called to get workspace origin.
◆ KeyDown (see page 513)	Called on key down in active designer.
◆ KeyPress (see page 513)	Called on key press in active designer.

### Legend

◆	Method
---	--------

#### IDesignIDEEvents Methods

IDesignIDEEvents Methods	Description
◆ ActiveDsnChanged (see page 512)	Called when active designer was changed.
◆ BeforeRegisterComponent (see page 512)	Called before registering component class.
◆ GetGlobalComponents (see page 512)	Called to get global components.
◆ GetWorkspaceOrigin (see page 513)	Called to get workspace origin.
◆ KeyDown (see page 513)	Called on key down in active designer.
◆ KeyPress (see page 513)	Called on key press in active designer.

## 1.43.2.2.1 IDesignIDEEvents Methods

### 1.43.2.2.1.1 IDesignIDEEvents.ActiveDsnChanged Method

Called when active designer was changed.

```
procedure ActiveDsnChanged;
```

### 1.43.2.2.1.2 IDesignIDEEvents.BeforeRegisterComponent Method

Called before registering component class.

```
procedure BeforeRegisterComponent(ComponentClass: TComponentClass; var Page: string; var
Accept: Boolean);
```

### 1.43.2.2.1.3 IDesignIDEEvents.GetGlobalComponents Method

Called to get global components.

```
function GetGlobalComponents(Root: TComponent; List: TList): Boolean;
```

### 1.43.2.2.1.4 IDesignIDEEvents.GetWorkspaceOrigin Method

Called to get workspace origin.

```
function GetWorkspaceOrigin(var Org: TPoint): Boolean;
```

### 1.43.2.2.1.5 IDesignIDEEvents.KeyDown Method

Called on key down in active designer.

```
procedure KeyDown(Sender: IDesigner; var Key: Word; Shift: TShiftState);
```

### 1.43.2.2.1.6 IDesignIDEEvents.KeyPress Method


Called on key press in active designer.

```
procedure KeyPress(Sender: IDesigner; var Key: Char);
```

## 1.43.3 Functions

The following table lists functions in this documentation.

### Functions

Function	Description
 GetClassDragImage (🔗 see page 513)	Returns image list with icon and optionally caption of selected component class.

### Legend

	Method
-------------------------------------------------------------------------------------	--------

### 1.43.3.1 edManager.GetClassDragImage Function

Returns image list with icon and optionally caption of selected component class.

```
function GetClassDragImage(WithCaption: Boolean): TImageList;
```


### File

edManager

## 1.43.4 Structs, Records, Enums

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

### Enumerations

Enumeration	Description
 TComponentClassDragImage (🔗 see page 513)	Kind of drag image for component palette.

### Legend

	Enumeration
-------------------------------------------------------------------------------------	-------------

### 1.43.4.1 edManager.TComponentClassDragImage Enumeration

Kind of drag image for component palette.

```
TComponentClassDragImage = (  
    cdiNone,  
    cdiIcon,  
    cdiIconAndCaption  
);
```

File

edManager


Members

Members	Description
cdiNone	No drag image.
cdiIcon	Displays only icon of component class.
cdiIconAndCaption	Displays icon and display name of component class.

# 1.43.5 Variables

The following table lists variables in this documentation.

Variables

Variable	Description
DsnManager  see page 514	Global designer event manager.

## 1.43.5.1 edManager.DsnManager Variable

Global designer event manager.

```
DsnManager: TDesignerManager;
```

File




edManager

# 1.44 edsMenuDsn Namespace

## 1.44.1 Classes

The following table lists classes in this documentation.

Classes

Class	Description
TMenuDsnWnd  see page 514	Use this dialog box to adjust menu items for TMainMenu and TPopupMenu components.
TzMenuEditor  see page 515	TzMenuEditor is a component editor for editing TMainMenu and TPopupMenu components.
TzMenuItemsPropertyEditor  see page 515	TzMenuItemsPropertyEditor is a string property editor especially designed to edit strings associated with menu item

### 1.44.1.1 TMenuDsnWnd Class

Use this dialog box to adjust menu items for TMainMenu and TPopupMenu components.

Class Hierarchy



```
TMenuDsnWnd = class(TDesignWindow);
```

**File**

edsMenuDsn

1.44.1.2 TzMenuEditor Class

TzMenuEditor is a component editor for editing TMainMenu and TPopupMenu components.

Class Hierarchy



```
TzMenuEditor = class(TDefaultEditor);
```

**File**

edsMenuDsn

**Description**

Add a description here...

1.44.1.3 TzMenuItemsPropertyEditor Class

TzMenuItemsPropertyEditor is a string property editor especially designed to edit strings associated with menu item

Class Hierarchy



```
TzMenuItemsPropertyEditor = class(TStringProperty);
```

**File**

edsMenuDsn

**Description**

1.45 eduDMContainer Namespace

1.45.1 Classes

The following table lists classes in this documentation.

Classes

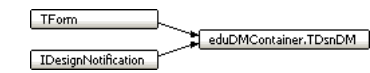
Class	Description
TDsnDM (🔗 see page 516)	TDsnDM is a form, designed to use as a DataModule container



### 1.45.1.1 TDsnDM Class

TDsnDM is a form, designed to use as a DataModule container

**Class Hierarchy**



```
TDsnDM = class(TForm, IDesignNotification);
```

**File**

eduDMContainer

**Description**

## 1.46 eduServObj Namespace

### 1.46.1 Classes

The following table lists classes in this documentation.

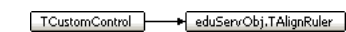
**Classes**

Class	Description
TAlignRuler (🔗 see page 516)	Popup window of 1 pixel width/height which is used as align ruler in the designer.
TComponentCaption (🔗 see page 517)	TComponentCaption is used as icon's Caption for nonvisual components
TComponentIcon (🔗 see page 517)	TComponentIcon is a class-wrapper to paint icons for non visual components.
TDraggedControl (🔗 see page 517)	Service window used during dragging controls (BDS Style).
TSmallRect (🔗 see page 517)	Implements single marker window.
TTabOrderIcons (🔗 see page 518)	Manages tab order icons.
TzBoundCtrl (🔗 see page 520)	Markers manager class.

### 1.46.1.1 TAlignRuler Class

Popup window of 1 pixel width/height which is used as align ruler in the designer.

**Class Hierarchy**



```
TAlignRuler = class(TCustomControl);
```

**File**

eduServObj

**Description**

### 1.46.1.2 TComponentCaption Class

TComponentCaption is used as icon's Caption for nonvisual components

#### Class Hierarchy



```
TComponentCaption = class(TCustomControl);
```

#### File

eduServObj

#### Description

### 1.46.1.3 TComponentIcon Class

TComponentIcon is a class-wrapper to paint icons for non visual components.

#### Class Hierarchy



```
TComponentIcon = class(TCustomControl);
```

#### File

eduServObj

### 1.46.1.4 TDraggedControl Class

Service window used during dragging controls (BDS Style).

#### Class Hierarchy



```
TDraggedControl = class(TCustomControl);
```

#### File

eduServObj

#### Description

### 1.46.1.5 TSmallRect Class

Implements single marker window.

#### Class Hierarchy



```
TSmallRect = class(TCustomControl);
```

#### File

eduServObj

**Description**

## 1.46.1.6 TTabOrderIcons Class

Manages tab order icons.

**Class Hierarchy**

```
TTabOrderIcons = class(TPersistent);
```

**File**

eduServObj

**Description**

"Show (see page 519) Tab Order" design mode allows easy changing of tab order of controls. This mode may be activated for any window control which has children controls.

In this mode over each child control with TabStop property equal to True icon is displayed with tab order index. Click on the control sequentially changes tab order index.

To exit "Show (see page 519) Tab Order" design mode click on any non-child control or press Escape key.

**Members****TTabOrderIcons Methods**

TTabOrderIcons Methods	Description
Hide (see page 519)	Exits "Show (see page 519) Tab Order" design mode.
SetTabOrder (see page 519)	Changes tab order of the specified child control.
Show (see page 519)	Activates "Show Tab Order" design mode for the specified parent control.

**TTabOrderIcons Properties**

TTabOrderIcons Properties	Description
Color (see page 519)	Specifies the color of tab order icon.
Font (see page 519)	Specifies font of the tab order icon.
Height (see page 519)	Specifies height of tab order icon. If it is equal to 0 - height is selected automatically to fit.
HorzAlign (see page 519)	Specifies horizontal position of tab order icon relative to associated control.
VertAlign (see page 519)	Specifies vertical position of tab order icon relative to associated control.
Visible (see page 519)	Returns True when "Show (see page 519) Tab Order" design mode is active.
Width (see page 520)	Specifies width of tab order icon. If it is equal to 0 - width is selected automatically to fit.

**Legend**







	Method
	Property
	read only

**TTabOrderIcons Methods**

TTabOrderIcons Methods	Description
Hide (see page 519)	Exits "Show (see page 519) Tab Order" design mode.
SetTabOrder (see page 519)	Changes tab order of the specified child control.
Show (see page 519)	Activates "Show Tab Order" design mode for the specified parent control.

**TTabOrderIcons Properties**

TTabOrderIcons Properties	Description
Color (see page 519)	Specifies the color of tab order icon.

 Font (see page 519)	Specifies font of the tab order icon.
 Height (see page 519)	Specifies height of tab order icon. If it is equal to 0 - height is selected automatically to fit.
 HorzAlign (see page 519)	Specifies horizontal position of tab order icon relative to associated control.
 VertAlign (see page 519)	Specifies vertical position of tab order icon relative to associated control.
 Visible (see page 519)	Returns True when "Show (see page 519) Tab Order" design mode is active.
 Width (see page 520)	Specifies width of tab order icon. If it is equal to 0 - width is selected automatically to fit.

## 1.46.1.6.1 TTabOrderIcons Methods

### 1.46.1.6.1.1 TTabOrderIcons.Hide Method

Exits "Show (see page 519) Tab Order" design mode.

```
procedure Hide;
```

### 1.46.1.6.1.2 TTabOrderIcons.SetTabOrder Method

Changes tab order of the specified child control.

```
function SetTabOrder(Ctl: TControl): Boolean;
```

### 1.46.1.6.1.3 TTabOrderIcons.Show Method

Activates "Show Tab Order" design mode for the specified parent control.

```
function Show(Prn: TWinControl): Boolean;
```

## 1.46.1.6.2 TTabOrderIcons Properties

### 1.46.1.6.2.1 TTabOrderIcons.Color Property

Specifies the color of tab order icon.

```
property Color: TColor;
```

### 1.46.1.6.2.2 TTabOrderIcons.Font Property

Specifies font of the tab order icon.

```
property Font: TFont;
```

### 1.46.1.6.2.3 TTabOrderIcons.Height Property

Specifies height of tab order icon. If it is equal to 0 - height is selected automatically to fit.

```
property Height: integer;
```

### 1.46.1.6.2.4 TTabOrderIcons.HorzAlign Property

Specifies horizontal position of tab order icon relative to associated control.

```
property HorzAlign: TAlignment;
```

### 1.46.1.6.2.5 TTabOrderIcons.VertAlign Property

Specifies vertical position of tab order icon relative to associated control.

```
property VertAlign: TVerticalAlignment;
```

### 1.46.1.6.2.6 TTabOrderIcons.Visible Property

Returns True when "Show (see page 519) Tab Order" design mode is active.

**property** Visible: boolean;

### 1.46.1.6.2.7 TTabOrderIcons.Width Property

Specifies width of tab order icon. If it is equal to 0 - width is selected automatically to fit.

**property** Width: integer;

## 1.46.1.7 TzBoundCtrl Class

Markers manager class.

### Class Hierarchy



```
TzBoundCtrl = class(TPersistent);
```

### File

eduServObj

### Description

Manages markers (up to 8) around selected in designer control.

### Members

#### TzBoundCtrl Methods

TzBoundCtrl Methods	Description
GrabAtPos ( see page 521)	Returns marker at specified position.
Hide ( see page 521)	Hides markers.
Invalidate ( see page 521)	Invalidates all markers.
Recreate ( see page 521)	Recreates markers.
Update ( see page 521)	Shows and moves markers to new control position.

#### TzBoundCtrl Properties

TzBoundCtrl Properties	Description
Bitmap ( see page 522)	Specifies bitmap of marker.
Color ( see page 522)	Indicates the color of the selections markers.
Control ( see page 522)	Specifies selected control.
DrawMultSel ( see page 522)	Specifies whether markers for multiple selection should be drawn.
GrabSize ( see page 522)	Indicates the size of the selections markers.
Local ( see page 522)	Specifies whether markers are visible only in the parent window of selected control. Otherwise markers are visible wherever in the form.
Locked ( see page 522)	Specifies locked state of markers.
MarkerShape ( see page 522)	Specifies shape of markers window.
Visible ( see page 523)	Specifies whether markers are visible.










### Legend

	Method
	protected
	Property

#### TzBoundCtrl Methods

TzBoundCtrl Methods	Description
GrabAtPos ( see page 521)	Returns marker at specified position.
Hide ( see page 521)	Hides markers.
Invalidate ( see page 521)	Invalidates all markers.
Recreate ( see page 521)	Recreates markers.
Update ( see page 521)	Shows and moves markers to new control position.

**TzBoundCtrl Properties**

<b>TzBoundCtrl Properties</b>	<b>Description</b>
 Bitmap (see page 522)	Specifies bitmap of marker.
 Color (see page 522)	Indicates the color of the selections markers.
 Control (see page 522)	Specifies selected control.
 DrawMultSel (see page 522)	Specifies whether markers for multiple selection should be drawn.
 GrabSize (see page 522)	Indicates the size of the selections markers.
 Local (see page 522)	Specifies whether markers are visible only in the parent window of selected control. Otherwise markers are visible wherever in the form.
 Locked (see page 522)	Specifies locked state of markers.
 MarkerShape (see page 522)	Specifies shape of markers window.
 Visible (see page 523)	Specifies whether markers are visible.

**1.46.1.7.1 TzBoundCtrl Methods****1.46.1.7.1.1 TzBoundCtrl.GrabAtPos Method**

Returns marker at specified position.

```
function GrabAtPos(p: TPoint): TSmallRect;
```

**Description****1.46.1.7.1.2 TzBoundCtrl.Hide Method**

Hides markers.

```
procedure Hide;
```

**Description****1.46.1.7.1.3 TzBoundCtrl.Invalidate Method**

Invalidates all markers.

```
procedure Invalidate;
```

**Description****1.46.1.7.1.4 TzBoundCtrl.Recreate Method**

Recreates markers.

```
procedure Recreate;
```

**1.46.1.7.1.5 TzBoundCtrl.Update Method**

Shows and moves markers to new control position.

```
procedure Update;
```

**Description****1.46.1.7.2 TzBoundCtrl Properties**

### 1.46.1.7.2.1 TzBoundCtrl.Bitmap Property

Specifies bitmap of marker.

```
property Bitmap: TBitmap;
```

#### Description

This bitmap should consist of two images in row: first for active state and second for locked state.

### 1.46.1.7.2.2 TzBoundCtrl.Color Property

Indicates the color of the selections markers.

```
property Color: TColor;
```

#### Description

Set this property to change color of the markers around selected components.

### 1.46.1.7.2.3 TzBoundCtrl.Control Property

Specifies selected control.

```
property Control: TControl;
```

#### Description

Markers are shown around selected control of the designer, when only one control is selected.

### 1.46.1.7.2.4 TzBoundCtrl.DrawMultSel Property

Specifies whether markers for multiple selection should be drawn.

```
property DrawMultSel: Boolean;
```

### 1.46.1.7.2.5 TzBoundCtrl.GrabSize Property

Indicates the size of the selections markers.

```
property GrabSize: integer;
```

#### Description

Set this property to change size of the markers around selected components.

### 1.46.1.7.2.6 TzBoundCtrl.Local Property

Specifies whether markers are visible only in the parent window of selected control. Otherwise markers are visible wherever in the form.

```
property Local: Boolean;
```

### 1.46.1.7.2.7 TzBoundCtrl.Locked Property

Specifies locked state of markers.

```
property Locked: Boolean;
```

#### Description

Markers are grayed in locked state (for example, when object can not be moved).

### 1.46.1.7.2.8 TzBoundCtrl.MarkerShape Property

Specifies shape of markers window.

```
property MarkerShape: TMarkerShape;
```

Description

1.46.1.7.2.9 TzBoundCtrl.Visible Property





Specifies whether markers are visible.

```
property Visible: Boolean;
```

1.46.2 Functions

The following table lists functions in this documentation.

Functions

Function	Description
 DrawPatternRect (  see page 523)	Draws pattern frame. Used by design surface to draw frame around designed form.
 IsServiceControl (  see page 523)	Returns true if Control is temporary service control managed by the designer.

Legend

	Method
-----------------------------------------------------------------------------------	--------

1.46.2.1 eduServObj.DrawPatternRect Function

```
procedure DrawPatternRect(Canvas: TCanvas; R: TRect; Size: integer);
```

File

eduServObj

Description

Draws pattern frame. Used by design surface to draw frame around designed form.

1.46.2.2 eduServObj.IsServiceControl Function

```
function IsServiceControl(AControl: TControl): Boolean;
```

File

eduServObj





Description

Returns true if Control is temporary service control managed by the designer.

1.46.3 Structs, Records, Enums

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

Enumerations

Enumeration	Description
 TMarkerShape (  see page 524)	Marker window shape
 TVerticalAlignment (  see page 524)	This is record eduServObj.TVerticalAlignment.



Legend

	Enumeration
-----------------------------------------------------------------------------------	-------------

1.46.3.1 eduServObj.TMarkerShape Enumeration

```
TMarkerShape = (  
    msReactangle,  
    msCircle,  
    msRoundRect  
);
```

File

eduServObj

Members

Members	Description
msReactangle	Rectangle shape
msCircle	Circle shape
msRoundRect	Rounded rectangle shape

Description

Marker window shape

1.46.3.2 eduServObj.TVerticalAlignment Enumeration

```
TVerticalAlignment = (  
    taAlignTop,  
    taAlignBottom,  
    taVerticalCenter  
);
```

File

eduServObj

Description





This is record eduServObj.TVerticalAlignment.




















1.47 edUtils Namespace

1.47.1 Functions

The following table lists functions in this documentation.

Functions

Function	Description
 DsnAlignSelected (see page 525)	Aligns selected objects in active designer.
 DsnLoadPackage (see page 525)	Loads package and registers design objects from it in the design environment.
 DsnReadCmpFromStream (see page 525)	Reads root component of Designer from Stream. Pass OnError and OnCreateCmp procedures to control loading process.
 DsnReadFromFile (see page 525)	Reads object (edited by Designer) resource from the file FileName. OnError allows handling of resource reading errors.

 <a href="#">DsnWriteCmpToStream</a> (  see page 526)	Write root component of Designer to stream. AsText parameter specifies stream format: text or binary.
 <a href="#">DsnWriteToFile</a> (  see page 526)	Writes edited by Designer object to the file FileName. AsText specifies format of written file.
 <a href="#">GetDesigner</a> (  see page 526)	Gets pointer to TzCustomFormDesigner (  see page 161) from Designer reference or from ActiveDesigner property in DsnManager if Designer = nil.
 <a href="#">InvalidateControl</a> (  see page 526)	Calls Invalidate for the specified control and all it's children
 <a href="#">IsControlParent</a> (  see page 526)	Validates whether AParent is a Parent of the AControl.
 <a href="#">NormalizeRect</a> (  see page 527)	Makes Left <= Right and Top <= Bottom
 <a href="#">PerformDsnAction</a> (  see page 527)	Performs designer action. If Designer = nil, active designer will be used.
 <a href="#">ShowDesignerOptionsDlg</a> (  see page 527)	Displays "Designer options" dialog. If Designer = nil, active designer will be used.
 <a href="#">ShowDsnAbout</a> (  see page 527)	Displays about dialog

**Legend**

	Method
-----------------------------------------------------------------------------------	--------

### 1.47.1.1 edUtils.DsnAlignSelected Function

```
procedure DsnAlignSelected(Horz: TCompAlign; Vert: TCompAlign);
```

**File**

edUtils

**Description**

Aligns selected objects in active designer.

### 1.47.1.2 edUtils.DsnLoadPackage Function

```
procedure DsnLoadPackage(const FileName: string);
```

**File**

edUtils

**Description**

Loads package and registers design objects from it in the design environment.

### 1.47.1.3 edUtils.DsnReadCmpFromStream Function

Reads root component of Designer from Stream. Pass OnError and OnCreateCmp procedures to control loading process.

```
procedure DsnReadCmpFromStream(Stream: TStream; const Designer: IDesigner; OnError: TReaderError = nil);
```

**File**

edUtils

### 1.47.1.4 edUtils.DsnReadFromFile Function

```
procedure DsnReadFromFile(const FileName: string; const Designer: IDesigner; OnError: TReaderError = nil);
```

**File**

edUtils

**Description**

Reads object (edited by Designer) resource from the file FileName. OnError allows handling of resource reading errors.

## 1.47.1.5 edUtils.DsnWriteCmpToStream Function

Write root component of Designer to stream. AsText parameter specifies stream format: text or binary.

```
procedure DsnWriteCmpToStream(Stream: TStream; const Designer: IDesigner; AsText: Boolean);
```

**File**

edUtils

## 1.47.1.6 edUtils.DsnWriteToFile Function

```
procedure DsnWriteToFile(const FileName: string; const Designer: IDesigner; AsText: Boolean);
```

**File**

edUtils

**Description**

Writes edited by Designer object to the file FileName. AsText specifies format of written file.

## 1.47.1.7 edUtils.GetDesigner Function

Gets pointer to TzCustomFormDesigner (see page 161) from Designer reference or from ActiveDesigner property in DsnManager if Designer = nil.

```
function GetDesigner(const Designer: IDesigner = nil): TzFormDesigner;
```

**File**

edUtils

**Description**

## 1.47.1.8 edUtils.InvalidateControl Function

```
procedure InvalidateControl(AControl: TControl);
```

**File**

edUtils

**Description**

Calls Invalidate for the specified control and all it's children

## 1.47.1.9 edUtils.IsControlParent Function

```
function IsControlParent(AControl: TControl; AParent: TWinControl): Boolean;
```

**File**

edUtils

**Description**

Validates whether AParent is a Parent of the AControl.

1.47.1.10 edUtils.NormalizeRect Function

```
procedure NormalizeRect(var Rect: TRect);
```

**File**

edUtils

**Description**

Makes Left <= Right and Top <= Bottom

1.47.1.11 edUtils.PerformDsnAction Function

```
function PerformDsnAction(act: TDesignerAction; const Designer: IDesigner = nil): Boolean;
```

**File**

edUtils

**Description**

Performs designer action. If Designer = nil, active designer will be used.

1.47.1.12 edUtils.ShowDesignerOptionsDlg Function

```
function ShowDesignerOptionsDlg(const Designer: IDesigner): Boolean;
```

**File**

edUtils

**Description**

Displays "Designer options" dialog. If Designer = nil, active designer will be used.

1.47.1.13 edUtils.ShowDsnAbout Function

```
procedure ShowDsnAbout;
```

**File**

edUtils


**Description**

Displays about dialog

1.47.2 Structs, Records, Enums

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

Enumerations

Enumeration	Description
 TDesignerAction (🔗 see page 528)	Designer specific actions.

Legend

	Enumeration
-----------------------------------------------------------------------------------	-------------

1.47.2.1 edUtils.TDesignerAction Enumeration

Designer specific actions.

```
TDesignerAction = (  
    daAlignToGrid,  
    daBringToFront,  
    daSendToBack,  
    daAlignmentDlg,  
    daSizeDlg,  
    daScale,  
    daTabOrderDlg,  
    daCreationOrderDlg,  
    daFlipChildrenAll,  
    daFlipChildren  
);
```

File

edUtils

Members

Members	Description
daAlignToGrid	Aligns selected controls to grid.
daBringToFront	Moves selected controls to front.
daSendToBack	Sends selected controls to back.
daAlignmentDlg	Displays "Alignment" dialog to align selected controls.
daSizeDlg	Displays "Size" dialog to change size of selected controls.
daScale	Display "Scale" dialog to resize controls.
daTabOrderDlg	Displays "Tab Order" dialog".
daCreationOrderDlg	Displays "Creation Order" dialog to change creation order of non-visible components.
daFlipChildrenAll	Flips all controls from left to right and vise versa.
daFlipChildren	Flips selected controls and their children.

# Index

## B

ButtonClick Example 10

## E

ecBtnPanel namespace

Classes 18

Structs, Records, Enums 44

Types 45

ecBtnPanel.TBtnMargins 18

ecBtnPanel.TBtnMargins.Bottom 19

ecBtnPanel.TBtnMargins.BtnHorz 19

ecBtnPanel.TBtnMargins.BtnVert 19

ecBtnPanel.TBtnMargins.Left 19

ecBtnPanel.TBtnMargins.Right 20

ecBtnPanel.TBtnMargins.Top 20

ecBtnPanel.TBtnPanel 28

ecBtnPanel.TBtnPanel.Align 32

ecBtnPanel.TBtnPanel.Anchors 33

ecBtnPanel.TBtnPanel.AutoSize 33

ecBtnPanel.TBtnPanel.BevelInner 33

ecBtnPanel.TBtnPanel.BevelOuter 33

ecBtnPanel.TBtnPanel.BevelWidth 34

ecBtnPanel.TBtnPanel.BiDiMode 34

ecBtnPanel.TBtnPanel.BorderStyle 34

ecBtnPanel.TBtnPanel.BorderWidth 34

ecBtnPanel.TBtnPanel.ButtonCount 35

ecBtnPanel.TBtnPanel.ButtonHeight 35

ecBtnPanel.TBtnPanel.ButtonWidth 35

ecBtnPanel.TBtnPanel.Color 35

ecBtnPanel.TBtnPanel.Constraints 36

ecBtnPanel.TBtnPanel.Ctl3D 36

ecBtnPanel.TBtnPanel.DownButton 36

ecBtnPanel.TBtnPanel.DragCursor 36

ecBtnPanel.TBtnPanel.DragKind 36

ecBtnPanel.TBtnPanel.DragMode 37

ecBtnPanel.TBtnPanel.Enabled 37

ecBtnPanel.TBtnPanel.Flat 37

ecBtnPanel.TBtnPanel.Font 37

ecBtnPanel.TBtnPanel.HintProps 37

ecBtnPanel.TBtnPanel.Margins 37

ecBtnPanel.TBtnPanel.OnButtonClick 38

ecBtnPanel.TBtnPanel.OnCanResize 38

ecBtnPanel.TBtnPanel.OnClick 38

ecBtnPanel.TBtnPanel.OnConstrainedResize 38

ecBtnPanel.TBtnPanel.OnContextPopup 39

ecBtnPanel.TBtnPanel.OnDbtClick 39

ecBtnPanel.TBtnPanel.OnDragDrop 39

ecBtnPanel.TBtnPanel.OnDragOver 39

ecBtnPanel.TBtnPanel.OnDrawButton 40

ecBtnPanel.TBtnPanel.OnEndDrag 40

ecBtnPanel.TBtnPanel.OnEnter 40

ecBtnPanel.TBtnPanel.OnExit 40

ecBtnPanel.TBtnPanel.OnGetButtonHint 41

ecBtnPanel.TBtnPanel.OnMouseDown 41

ecBtnPanel.TBtnPanel.OnMouseMove 41

ecBtnPanel.TBtnPanel.OnMouseUp 41

ecBtnPanel.TBtnPanel.OnResize 42

ecBtnPanel.TBtnPanel.OnStartDrag 42

ecBtnPanel.TBtnPanel.Orientation 42

ecBtnPanel.TBtnPanel.ParentBiDiMode 42

ecBtnPanel.TBtnPanel.ParentColor 42

ecBtnPanel.TBtnPanel.ParentCtl3D 43

ecBtnPanel.TBtnPanel.ParentFont 43

ecBtnPanel.TBtnPanel.ParentShowHint 43

ecBtnPanel.TBtnPanel.PopupMenu 43

ecBtnPanel.TBtnPanel.RowCount 43

ecBtnPanel.TBtnPanel.ShowHint 43

ecBtnPanel.TBtnPanel.TabOrder 43

ecBtnPanel.TBtnPanel.TabStop 43

ecBtnPanel.TBtnPanel.Transparent 44

ecBtnPanel.TBtnPanel.Visible 44

ecBtnPanel.TButtonClickEvent 45

ecBtnPanel.TButtonClickEvent type 45

ecBtnPanel.TCustomBtnPanel 20

ecBtnPanel.TCustomBtnPanel.AutoSize 25

ecBtnPanel.TCustomBtnPanel.ButtonAtPos 22

ecBtnPanel.TCustomBtnPanel.ButtonClick 22

ecBtnPanel.TCustomBtnPanel.ButtonCount 25

ecBtnPanel.TCustomBtnPanel.ButtonHeight 25

ecBtnPanel.TCustomBtnPanel.ButtonRect 22

ecBtnPanel.TCustomBtnPanel.ButtonWidth 25

---

ecBtnPanel.TCustomBtnPanel.CanAutoSize 23	ecDList.TCustomPropList.Destroy 49
ecBtnPanel.TCustomBtnPanel.Caption 25	ecDList.TCustomPropList.DoPrepareCanvas 50
ecBtnPanel.TCustomBtnPanel.Create 23	ecDList.TCustomPropList.DrawCell 50
ecBtnPanel.TCustomBtnPanel.Destroy 23	ecDList.TCustomPropList.DrawPropCell 50
ecBtnPanel.TCustomBtnPanel.DownButton 26	ecDList.TCustomPropList.FoldingIcon 51
ecBtnPanel.TCustomBtnPanel.DrawButton 23	ecDList.TCustomPropList.GutterWidth 50
ecBtnPanel.TCustomBtnPanel.Flat 26	ecDList.TCustomPropList.IsHeaderItem 50
ecBtnPanel.TCustomBtnPanel.GetButtonHint 24	ecDList.TCustomPropList.Items 51
ecBtnPanel.TCustomBtnPanel.HintProps 26	ecDList.TCustomPropList.LeftMargin 51
ecBtnPanel.TCustomBtnPanel.InvalidateButtons 24	ecDList.TCustomPropList.LevelWidth 51
ecBtnPanel.TCustomBtnPanel.Loaded 24	ecDList.TCustomPropList.MouseDown 50
ecBtnPanel.TCustomBtnPanel.Margins 26	ecDList.TCustomPropList.OnDrawPropCell 51
ecBtnPanel.TCustomBtnPanel.MouseDown 24	ecDList.TCustomPropList.OnGetCellParams 52
ecBtnPanel.TCustomBtnPanel.OnButtonClick 27	ecDList.TCustomPropList.ShowGutter 51
ecBtnPanel.TCustomBtnPanel.OnDrawButton 27	ecDList.TDualList 52
ecBtnPanel.TCustomBtnPanel.OnGetButtonHint 28	ecDList.TDualList.BorderStyle 58
ecBtnPanel.TCustomBtnPanel.Orientation 26	ecDList.TDualList.Canvas 58
ecBtnPanel.TCustomBtnPanel.Paint 24	ecDList.TDualList.Create 54
ecBtnPanel.TCustomBtnPanel.RowCount 27	ecDList.TDualList.CreateEditor 54
ecBtnPanel.TCustomBtnPanel.Transparent 27	ecDList.TDualList.CreateHandle 54
ecBtnPanel.TDrawButtonEvent 45	ecDList.TDualList.Destroy 54
ecBtnPanel.TDrawButtonEvent type 45	ecDList.TDualList.DoMouseWheel 54
ecBtnPanel.TGetButtonHintEvent 45	ecDList.TDualList.DrawCell 55
ecBtnPanel.TGetButtonHintEvent type 45	ecDList.TDualList.DrawStr 55
ecBtnPanel.TRowOrientation 44	ecDList.TDualList.DrawStrW 55
ecBtnPanel.TRowOrientation enumeration 44	ecDList.TDualList.Editor 59
ecDList namespace	ecDList.TDualList.EditorVisible 59
Classes 46	ecDList.TDualList.ExecuteAction 55
Structs, Records, Enums 69	ecDList.TDualList.FocusEditor 56
Types 69	ecDList.TDualList.IsHeaderItem 56
ecDList.TCellType 69	ecDList.TDualList.ItemCount 59
ecDList.TCellType enumeration 69	ecDList.TDualList.ItemHeight 59
ecDList.TCustomPropDrawEvent 70	ecDList.TDualList.ItemIndex 59
ecDList.TCustomPropDrawEvent type 70	ecDList.TDualList.ItemRect 56
ecDList.TCustomPropList 46	ecDList.TDualList.KeyDown 56
ecDList.TCustomPropList.cGutter 50	ecDList.TDualList.MouseDown 56
ecDList.TCustomPropList.cGutterBnd 51	ecDList.TDualList.MouseMove 56
ecDList.TCustomPropList.cHighlight 51	ecDList.TDualList.MouseToItem 57
ecDList.TCustomPropList.cHighlightText 51	ecDList.TDualList.MouseUp 57
ecDList.TCustomPropList.Create 49	ecDList.TDualList.OnClick 60
ecDList.TCustomPropList.CreateItems 49	ecDList.TDualList.Paint 57
ecDList.TCustomPropList.Current 49	ecDList.TDualList.SetItemIndex 57

---

---

ecDList.TDualList.ShowGrid 60	ecDList.TPropListRoot.ExpCount 68
ecDList.TDualList.ShowSelFrame 60	ecDList.TPropListRoot.ExplIndexOf 68
ecDList.TDualList.SplitPos 60	ecDList.TPropListRoot.ExplItems 68
ecDList.TDualList.TabOrder 60	ecDList.TPropListRoot.Owner 69
ecDList.TDualList.TabStop 60	ecDList.TPropListRoot.RestoreState 68
ecDList.TDualList.TopItem 60	ecDList.TPropListRoot.SaveState 68
ecDList.TDualList.UpdateAction 57	ecDList.TPropListRoot.UpdateList 68
ecDList.TDualList.UpdateEditor 58	ecExtEdit namespace
ecDList.TGetCellParamsEvent 70	Classes 70
ecDList.TGetCellParamsEvent type 70	Structs, Records, Enums 109
ecDList.TPropertyItem 61	Types 110
ecDList.TPropertyItem.Add 62	ecExtEdit.TBtnEdit 70
ecDList.TPropertyItem.Changed 62	ecExtEdit.TBtnEdit.AdjustClientRect 73
ecDList.TPropertyItem.Clear 62	ecExtEdit.TBtnEdit.Alignment 76
ecDList.TPropertyItem.Count 64	ecExtEdit.TBtnEdit.ButtonClick 73
ecDList.TPropertyItem.Create 62	ecExtEdit.TBtnEdit.ButtonVisible 76
ecDList.TPropertyItem.Delete 63	ecExtEdit.TBtnEdit.ButtonWidth 77
ecDList.TPropertyItem.Destroy 63	ecExtEdit.TBtnEdit.Canvas 77
ecDList.TPropertyItem.DisplayName 64	ecExtEdit.TBtnEdit.Create 73
ecDList.TPropertyItem.Expandable 63	ecExtEdit.TBtnEdit.CreateParams 73
ecDList.TPropertyItem.Expanded 64	ecExtEdit.TBtnEdit.CreateWnd 73
ecDList.TPropertyItem.GetName 63	ecExtEdit.TBtnEdit.Destroy 74
ecDList.TPropertyItem.HasValue 63	ecExtEdit.TBtnEdit.EndTracking 74
ecDList.TPropertyItem.IndexOf 63	ecExtEdit.TBtnEdit.KeyDown 74
ecDList.TPropertyItem.Insert 63	ecExtEdit.TBtnEdit.KeyPress 74
ecDList.TPropertyItem.IsEqual 63	ecExtEdit.TBtnEdit.MouseMove 74
ecDList.TPropertyItem.IsRoot 63	ecExtEdit.TBtnEdit.MouseUp 75
ecDList.TPropertyItem.Items 64	ecExtEdit.TBtnEdit.MultiLine 77
ecDList.TPropertyItem.Level 64	ecExtEdit.TBtnEdit.OnButtonClick 78
ecDList.TPropertyItem.Move 63	ecExtEdit.TBtnEdit.Paint 75
ecDList.TPropertyItem.Name 64	ecExtEdit.TBtnEdit.PaintBtnGlyph 75
ecDList.TPropertyItem.Parent 65	ecExtEdit.TBtnEdit.PaintStatus 75
ecDList.TPropertyItem.PathName 65	ecExtEdit.TBtnEdit.PaintWindow 75
ecDList.TPropertyItem.Root 64	ecExtEdit.TBtnEdit.PtInButton 75
ecDList.TPropertyItem.Visible 65	ecExtEdit.TBtnEdit.StartTracking 76
ecDList.TPropListRoot 65	ecExtEdit.TBtnEdit.StatusWidth 77
ecDList.TPropListRoot.BeginUpdate 67	ecExtEdit.TBtnEdit.StopTracking 76
ecDList.TPropListRoot.Changed 67	ecExtEdit.TBtnEdit.TrackButton 76
ecDList.TPropListRoot.Create 67	ecExtEdit.TBtnEdit.WantReturns 77
ecDList.TPropListRoot.Destroy 68	ecExtEdit.TBtnEdit.WantTabs 78
ecDList.TPropListRoot.EndUpdate 68	ecExtEdit.TBtnEdit.WordWrap 78
ecDList.TPropListRoot.ExpandItem 68	ecExtEdit.TCloseUpEvent 110

---



ecExtEdit.TCloseUpEvent type 110	ecExtEdit.TEditEx.DragMode 95
ecExtEdit.TCustomEditEx 78	ecExtEdit.TEditEx.EditMask 96
ecExtEdit.TCustomEditEx.AcceptListValue 82	ecExtEdit.TEditEx.EditStyle 96
ecExtEdit.TCustomEditEx.ActiveList 84	ecExtEdit.TEditEx.Enabled 96
ecExtEdit.TCustomEditEx.ButtonClick 82	ecExtEdit.TEditEx.Font 96
ecExtEdit.TCustomEditEx.CloseUp 82	ecExtEdit.TEditEx.ImeMode 96
ecExtEdit.TCustomEditEx.Create 82	ecExtEdit.TEditEx.ImeName 97
ecExtEdit.TCustomEditEx.Destroy 82	ecExtEdit.TEditEx.IsUnicode 97
ecExtEdit.TCustomEditEx.DoDropDownKeys 83	ecExtEdit.TEditEx.ListAlign 97
ecExtEdit.TCustomEditEx.DropDown 83	ecExtEdit.TEditEx.MaxLength 97
ecExtEdit.TCustomEditEx.EditStyle 84	ecExtEdit.TEditEx.OnAcceptListValue 98
ecExtEdit.TCustomEditEx.EndTracking 83	ecExtEdit.TEditEx.OnButtonClick 98
ecExtEdit.TCustomEditEx.KeyPress 83	ecExtEdit.TEditEx.OnChange 98
ecExtEdit.TCustomEditEx.ListAlign 84	ecExtEdit.TEditEx.OnClick 98
ecExtEdit.TCustomEditEx.MouseDown 83	ecExtEdit.TEditEx.OnCloseUp 98
ecExtEdit.TCustomEditEx.MouseMove 83	ecExtEdit.TEditEx.OnDbClick 98
ecExtEdit.TCustomEditEx.OnAcceptListValue 85	ecExtEdit.TEditEx.OnDragDrop 98
ecExtEdit.TCustomEditEx.OnCloseUp 85	ecExtEdit.TEditEx.OnDragOver 99
ecExtEdit.TCustomEditEx.OnDropDown 85	ecExtEdit.TEditEx.OnDropDown 99
ecExtEdit.TCustomEditEx.OnMeasureWidth 85	ecExtEdit.TEditEx.OnEndDrag 99
ecExtEdit.TCustomEditEx.PaintBtnGlyph 84	ecExtEdit.TEditEx.OnEnter 99
ecExtEdit.TCustomEditEx.PickList 84	ecExtEdit.TEditEx.OnExit 100
ecExtEdit.TCustomEditEx.StartTracking 84	ecExtEdit.TEditEx.OnKeyDown 100
ecExtEdit.TEditEx 85	ecExtEdit.TEditEx.OnKeyPress 100
ecExtEdit.TEditEx.Alignment 92	ecExtEdit.TEditEx.OnKeyUp 100
ecExtEdit.TEditEx.Anchors 92	ecExtEdit.TEditEx.OnMeasureWidth 101
ecExtEdit.TEditEx.AutoSelect 92	ecExtEdit.TEditEx.OnMouseDown 101
ecExtEdit.TEditEx.AutoSize 92	ecExtEdit.TEditEx.OnMouseMove 101
ecExtEdit.TEditEx.BevelEdges 93	ecExtEdit.TEditEx.OnMouseUp 101
ecExtEdit.TEditEx.BevelInner 93	ecExtEdit.TEditEx.OnStartDrag 102
ecExtEdit.TEditEx.BevelKind 93	ecExtEdit.TEditEx.ParentBiDiMode 102
ecExtEdit.TEditEx.BevelOuter 93	ecExtEdit.TEditEx.ParentColor 102
ecExtEdit.TEditEx.BevelWidth 93	ecExtEdit.TEditEx.ParentCtl3D 102
ecExtEdit.TEditEx.BiDiMode 94	ecExtEdit.TEditEx.ParentFont 102
ecExtEdit.TEditEx.BorderStyle 94	ecExtEdit.TEditEx.ParentShowHint 102
ecExtEdit.TEditEx.ButtonWidth 94	ecExtEdit.TEditEx.PasswordChar 103
ecExtEdit.TEditEx.CharCase 94	ecExtEdit.TEditEx.PickList 103
ecExtEdit.TEditEx.Color 95	ecExtEdit.TEditEx.PopupMenu 103
ecExtEdit.TEditEx.Constraints 95	ecExtEdit.TEditEx.ReadOnly 103
ecExtEdit.TEditEx.Ctl3D 95	ecExtEdit.TEditEx SelTextW 103
ecExtEdit.TEditEx.DragCursor 95	ecExtEdit.TEditEx.ShowHint 104
ecExtEdit.TEditEx.DragKind 95	ecExtEdit.TEditEx.StatusWidth 104

- ecExtEdit.TEditEx.TabOrder 104
- ecExtEdit.TEditEx.TabStop 104
- ecExtEdit.TEditEx.Text 104
- ecExtEdit.TEditEx.TextW 104
- ecExtEdit.TEditEx.Visible 104
- ecExtEdit.TInplaceEditStyle 110
- ecExtEdit.TInplaceEditStyle enumeration 110
- ecExtEdit.TMeasureWidthEvent 110
- ecExtEdit.TMeasureWidthEvent type 110
- ecExtEdit.TOnAcceptListValueEvent 111
- ecExtEdit.TOnAcceptListValueEvent type 111
- ecExtEdit.TPopupListbox 105
- ecExtEdit.TPopupListbox.CreateParams 106
- ecExtEdit.TPopupListbox.CreateWnd 106
- ecExtEdit.TPopupListbox.ItemHeight 106
- ecExtEdit.TPopupListbox.KeyPress 106
- ecExtEdit.TPopupListbox.OnDrawItem 106
- ecExtEdit.TPopupListbox.OnMeasureItem 107
- ecExtEdit.TPopupListbox.Sorted 107
- ecExtEdit.TPopupListbox.Style 107
- ecExtEdit.TUnicodeEdit 107
- ecExtEdit.TUnicodeEdit.Create 108
- ecExtEdit.TUnicodeEdit.Destroy 108
- ecExtEdit.TUnicodeEdit.IsUnicode 108
- ecExtEdit.TUnicodeEdit.SelTextW 109
- ecExtEdit.TUnicodeEdit.Text 109
- ecExtEdit.TUnicodeEdit.TextW 109
- ecHintHelper namespace
  - Classes 111
  - Constants 115
  - Structs, Records, Enums 114
  - Types 115
- ecHintHelper.CM\_GETHINTDATA 116
- ecHintHelper.CM\_GETHINTDATA constant 116
- ecHintHelper.PecHintData 115
- ecHintHelper.PecHintData type 115
- ecHintHelper.TCMGetHintData 114
- ecHintHelper.TCMGetHintData record 114
- ecHintHelper.TechHintData 115
- ecHintHelper.TechHintData record 115
- ecHintHelper.TechHintHelper 111
- ecHintHelper.TechHintHelper.CancelHint 112
- ecHintHelper.TechHintHelper.CanMoveLeft 113
- ecHintHelper.TechHintHelper.Color 113
- ecHintHelper.TechHintHelper.ControlWndProc 112
- ecHintHelper.TechHintHelper.Create 112
- ecHintHelper.TechHintHelper.Destroy 113
- ecHintHelper.TechHintHelper.Enabled 113
- ecHintHelper.TechHintHelper.Font 113
- ecHintHelper.TechHintHelper.HidePause 114
- ecHintHelper.TechHintHelper.Pause 114
- ecHintHelper.TechHintHelper.ResetHint 113
- ecHintHelper.TechHintHelper.ShortPause 114
- ecHintHelper.TechHintHelper.ShowHint 113
- ecToolList namespace
  - Classes 116
  - Structs, Records, Enums 143
  - Types 144
- ecToolList.TCustomToolList 116
- ecToolList.TCustomToolList.AllowArrange 120
- ecToolList.TCustomToolList.AutoCollapse 120
- ecToolList.TCustomToolList.CategoryHeight 120
- ecToolList.TCustomToolList.CollapseAll 118
- ecToolList.TCustomToolList.Create 119
- ecToolList.TCustomToolList.Destroy 119
- ecToolList.TCustomToolList.DrawItemImage 119
- ecToolList.TCustomToolList.ExpandAll 119
- ecToolList.TCustomToolList.Filtered 120
- ecToolList.TCustomToolList.FilterString 121
- ecToolList.TCustomToolList.FoldingIcon 121
- ecToolList.TCustomToolList.GetCategoryItem 119
- ecToolList.TCustomToolList.HintProps 121
- ecToolList.TCustomToolList.Images 121
- ecToolList.TCustomToolList.InsertAtItem 121
- ecToolList.TCustomToolList.ItemAtPos 119
- ecToolList.TCustomToolList.ItemHeight 121
- ecToolList.TCustomToolList.ItemIndex 121
- ecToolList.TCustomToolList.ItemIndexChanged 119
- ecToolList.TCustomToolList.ItemRect 119
- ecToolList.TCustomToolList.Items 121
- ecToolList.TCustomToolList.ItemsArranged 120
- ecToolList.TCustomToolList.ItemsChanged 120
- ecToolList.TCustomToolList.ItemsHeight 120
- ecToolList.TCustomToolList.MakeTopItem 120

---

ecToolList.TCustomToolList.MakeVisible 120	ecToolList.TToolList.Color 132
ecToolList.TCustomToolList.MouseOverItem 122	ecToolList.TToolList.Constraints 133
ecToolList.TCustomToolList.OnItemArranged 123	ecToolList.TToolList.Ctl3D 133
ecToolList.TCustomToolList.OnItemChanged 123	ecToolList.TToolList.DragCursor 133
ecToolList.TCustomToolList.PaintItem 120	ecToolList.TToolList.DragKind 133
ecToolList.TCustomToolList.RightClickSelect 122	ecToolList.TToolList.DragMode 133
ecToolList.TCustomToolList.RowSpace 122	ecToolList.TToolList.Enabled 133
ecToolList.TCustomToolList.Selected 122	ecToolList.TToolList.Filtered 134
ecToolList.TCustomToolList.SelectFirstVisible 120	ecToolList.TToolList.FilterString 134
ecToolList.TCustomToolList.StyleCategory 122	ecToolList.TToolList.FoldingIcon 134
ecToolList.TCustomToolList.StyleCategoryMouseOver 122	ecToolList.TToolList.Font 134
ecToolList.TCustomToolList.StyleCategorySelected 122	ecToolList.TToolList.HintProps 134
ecToolList.TCustomToolList.StyleItem 122	ecToolList.TToolList.Images 134
ecToolList.TCustomToolList.StyleItemMouseOver 122	ecToolList.TToolList.ItemHeight 134
ecToolList.TCustomToolList.StyleItemSelected 122	ecToolList.TToolList.ItemIndex 135
ecToolList.TCustomToolList.VerticalGroups 122	ecToolList.TToolList.Items 135
ecToolList.TCustomToolList.ViewOrigin 123	ecToolList.TToolList.OnCanResize 135
ecToolList.TItemShape 144	ecToolList.TToolList.OnClick 135
ecToolList.TItemShape enumeration 144	ecToolList.TToolList.OnConstrainedResize 135
ecToolList.TToolItemState 144	ecToolList.TToolList.OnContextPopup 136
ecToolList.TToolItemState type 144	ecToolList.TToolList.OnDbClick 136
ecToolList.TToolItemStyle 123	ecToolList.TToolList.OnDragDrop 136
ecToolList.TToolItemStyle.Alignment 124	ecToolList.TToolList.OnDragOver 136
ecToolList.TToolItemStyle.BoundPen 125	ecToolList.TToolList.OnEndDrag 137
ecToolList.TToolItemStyle.Brush 125	ecToolList.TToolList.OnEnter 137
ecToolList.TToolItemStyle.Create 124	ecToolList.TToolList.OnExit 137
ecToolList.TToolItemStyle.Destroy 124	ecToolList.TToolList.OnItemArranged 137
ecToolList.TToolItemStyle.DrawItemRect 124	ecToolList.TToolList.OnItemChanged 137
ecToolList.TToolItemStyle.Font 125	ecToolList.TToolList.OnMouseDown 137
ecToolList.TToolItemStyle.OnChange 125	ecToolList.TToolList.OnMouseMove 138
ecToolList.TToolItemStyle.Shape 125	ecToolList.TToolList.OnMouseUp 138
ecToolList.TToolList 125	ecToolList.TToolList.OnResize 138
ecToolList.TToolList.Align 130	ecToolList.TToolList.OnStartDrag 138
ecToolList.TToolList.AllowArrange 131	ecToolList.TToolList.ParentBiDiMode 139
ecToolList.TToolList.Anchors 131	ecToolList.TToolList.ParentColor 139
ecToolList.TToolList.AutoCollapse 131	ecToolList.TToolList.ParentCtl3D 139
ecToolList.TToolList.BevelEdges 131	ecToolList.TToolList.ParentFont 139
ecToolList.TToolList.BevelInner 131	ecToolList.TToolList.ParentShowHint 139
ecToolList.TToolList.BevelKind 132	ecToolList.TToolList.PopupMenu 139
ecToolList.TToolList.BevelOuter 132	ecToolList.TToolList.RightClickSelect 139
ecToolList.TToolList.BiDiMode 132	ecToolList.TToolList.RowSpace 140
ecToolList.TToolList.CategoryHeight 132	ecToolList.TToolList.Selected 140

---

---

ecToolList.TToolList.ShowHint 140  
 ecToolList.TToolList.StyleCategory 140  
 ecToolList.TToolList.StyleCategoryMouseOver 140  
 ecToolList.TToolList.StyleCategorySelected 140  
 ecToolList.TToolList.StyleItem 140  
 ecToolList.TToolList.StyleItemMouseOver 140  
 ecToolList.TToolList.StyleItemSelected 140  
 ecToolList.TToolList.TabOrder 140  
 ecToolList.TToolList.TabStop 140  
 ecToolList.TToolList.VerticalGroups 141  
 ecToolList.TToolList.Visible 141  
 ecToolList.TToolListItem 141  
 ecToolList.TToolListItem.Caption 142  
 ecToolList.TToolListItem.Expanded 142  
 ecToolList.TToolListItem.Hint 142  
 ecToolList.TToolListItem.ImageIndex 142  
 ecToolList.TToolListItem.IsCategory 142  
 ecToolList.TToolListItem.Tag 142  
 ecToolList.TToolListItem.ToolList 143  
 ecToolList.TToolListItem.Visible 143  
 ecToolList.TToolListItems 143  
 ecToolList.TToolListItems.Items 143  
 ed\_Designer namespace  
     Classes 156  
     Constants 236  
     Structs, Records, Enums 229  
     Types 231  
 ed\_Designer.DM\_POSCHANGED 236  
 ed\_Designer.DM\_POSCHANGED constant 236  
 ed\_Designer.sLineBreak 236  
 ed\_Designer.sLineBreak constant 236  
 ed\_Designer.TBufferizedType 229  
 ed\_Designer.TBufferizedType enumeration 229  
 ed\_Designer.TCompAlign 229  
 ed\_Designer.TCompAlign enumeration 229  
 ed\_Designer.TComponentEvent 232  
 ed\_Designer.TComponentEvent type 232  
 ed\_Designer.TCompSize 230  
 ed\_Designer.TCompSize enumeration 230  
 ed\_Designer.TControlGroups 156  
 ed\_Designer.TControlGroups.Clear 158  
 ed\_Designer.TControlGroups.Count 158  
 ed\_Designer.TControlGroups.Create 158  
 ed\_Designer.TControlGroups.Destroy 158  
 ed\_Designer.TControlGroups.GroupControls 158  
 ed\_Designer.TControlGroups.GroupForControl 158  
 ed\_Designer.TControlGroups.Groups 158  
 ed\_Designer.TControlGroups.GroupSelected 158  
 ed\_Designer.TControlGroups.UnGroup 158  
 ed\_Designer.TControlGroups.UnGroupSelected 158  
 ed\_Designer.TCreateComponentEvent 232  
 ed\_Designer.TCreateComponentEvent type 232  
 ed\_Designer.TCreateFrameEvent 232  
 ed\_Designer.TCreateFrameEvent type 232  
 ed\_Designer.TCreateIconEvent 232  
 ed\_Designer.TCreateIconEvent type 232  
 ed\_Designer.TCreateMethodEvent 233  
 ed\_Designer.TCreateMethodEvent type 233  
 ed\_Designer.TDrawControlEvent 233  
 ed\_Designer.TDrawControlEvent type 233  
 ed\_Designer.TGetComponentHintEvent 233  
 ed\_Designer.TGetComponentHintEvent type 233  
 ed\_Designer.TGetMethodNamesEvent 233  
 ed\_Designer.TGetMethodNamesEvent type 233  
 ed\_Designer.TGetObjNameEvent 233  
 ed\_Designer.TGetObjNameEvent type 233  
 ed\_Designer.TGetScriptProcEvent 234  
 ed\_Designer.TGetScriptProcEvent type 234  
 ed\_Designer.TGuidelinesStyle 230  
 ed\_Designer.TGuidelinesStyle enumeration 230  
 ed\_Designer.TGuidelinesStyles 234  
 ed\_Designer.TGuidelinesStyles type 234  
 ed\_Designer.THandleActionEvent 234  
 ed\_Designer.THandleActionEvent type 234  
 ed\_Designer.TLocalMenuFilter 231  
 ed\_Designer.TLocalMenuFilter enumeration 231  
 ed\_Designer.TLocalMenuFilters 234  
 ed\_Designer.TLocalMenuFilters type 234  
 ed\_Designer.TNotificationEvent 234  
 ed\_Designer.TNotificationEvent type 234  
 ed\_Designer.TPasteInfo 159  
 ed\_Designer.TPasteInfo.Create 159  
 ed\_Designer.TPasteInfo.CurrOffset 160  
 ed\_Designer.TPasteInfo.Destroy 160

---

---

ed_Designer.TPasteInfo.IncForParent 160	ed_Designer.TzCustomFormDesigner.CloseTextEditor 171
ed_Designer.TPasteInfo.Init 160	ed_Designer.TzCustomFormDesigner.ContainerWindow 188
ed_Designer.TRenameEvent 235	ed_Designer.TzCustomFormDesigner.CopySelection 174
ed_Designer.TRenameEvent type 235	ed_Designer.TzCustomFormDesigner.Create 174
ed_Designer.TRenameMethodEvent 235	ed_Designer.TzCustomFormDesigner.CutSelection 174
ed_Designer.TRenameMethodEvent type 235	ed_Designer.TzCustomFormDesigner.DeleteSelection 175
ed_Designer.TSetNameEvent 235	ed_Designer.TzCustomFormDesigner.DesignSurface 188
ed_Designer.TSetNameEvent type 235	ed_Designer.TzCustomFormDesigner.Destroy 175
ed_Designer.TSetScriptProcEvent 235	ed_Designer.TzCustomFormDesigner.DisplayControlGrid 189
ed_Designer.TSetScriptProcEvent type 235	ed_Designer.TzCustomFormDesigner.DisplayGrid 189
ed_Designer.TShowMethodEvent 235	ed_Designer.TzCustomFormDesigner.DoObjectHint 175
ed_Designer.TShowMethodEvent type 235	ed_Designer.TzCustomFormDesigner.DragDraw 175
ed_Designer.TUndoRecEvent 236	ed_Designer.TzCustomFormDesigner.DragDrop 171
ed_Designer.TUndoRecEvent type 236	ed_Designer.TzCustomFormDesigner.DragOver 171
ed_Designer.TValidateMethodEvent 236	ed_Designer.TzCustomFormDesigner.DragParentLimit 189
ed_Designer.TValidateMethodEvent type 236	ed_Designer.TzCustomFormDesigner.Edit 175
ed_Designer.TzCustomFormDesigner 161	ed_Designer.TzCustomFormDesigner.EditAction 172
ed_Designer.TzCustomFormDesigner.AddCompEditorMenu 170	ed_Designer.TzCustomFormDesigner.EndDrag 176
ed_Designer.TzCustomFormDesigner.AlignSelected 172	ed_Designer.TzCustomFormDesigner.Events 189
ed_Designer.TzCustomFormDesigner.AlignToGrid 172	ed_Designer.TzCustomFormDesigner.ExecuteAction 175
ed_Designer.TzCustomFormDesigner.AllowComponents 187	ed_Designer.TzCustomFormDesigner.FlatIcons 189
ed_Designer.TzCustomFormDesigner.AutoAlign 187	ed_Designer.TzCustomFormDesigner.FlipChildren 176
ed_Designer.TzCustomFormDesigner.BDSStyle 188	ed_Designer.TzCustomFormDesigner.Form 189
ed_Designer.TzCustomFormDesigner.BringToFront 172	ed_Designer.TzCustomFormDesigner.GetCompObj 176
ed_Designer.TzCustomFormDesigner.BuildLocalMenu 172	ed_Designer.TzCustomFormDesigner.GetComponent 176
ed_Designer.TzCustomFormDesigner.CancelDrag 173	ed_Designer.TzCustomFormDesigner.GetComponentName 176
ed_Designer.TzCustomFormDesigner.CanDelete 173	ed_Designer.TzCustomFormDesigner.GetComponentNames 177
ed_Designer.TzCustomFormDesigner.CanInsert 173	ed_Designer.TzCustomFormDesigner.GetControlAt 177
ed_Designer.TzCustomFormDesigner.CanMove 173	ed_Designer.TzCustomFormDesigner.GetEditState 173
ed_Designer.TzCustomFormDesigner.CanPaste 173	ed_Designer.TzCustomFormDesigner.GetMethodName 177
ed_Designer.TzCustomFormDesigner.CanRedo 170	ed_Designer.TzCustomFormDesigner.GetNewName 177
ed_Designer.TzCustomFormDesigner.CanRename 173	ed_Designer.TzCustomFormDesigner.GetObjectName 177
ed_Designer.TzCustomFormDesigner.CanResize 174	ed_Designer.TzCustomFormDesigner.GetRoot 178
ed_Designer.TzCustomFormDesigner.CanSelect 174	ed_Designer.TzCustomFormDesigner.GetRootClassName 178
ed_Designer.TzCustomFormDesigner.CanUndo 171	ed_Designer.TzCustomFormDesigner.GetScriptEvent 178
ed_Designer.TzCustomFormDesigner.CaptionFont 188	ed_Designer.TzCustomFormDesigner.GetScrollRanges 178
ed_Designer.TzCustomFormDesigner.CheckAction 171	ed_Designer.TzCustomFormDesigner.GetSelections 178
ed_Designer.TzCustomFormDesigner.ClearCompEditorMenu 171	ed_Designer.TzCustomFormDesigner.GetShiftState 179
ed_Designer.TzCustomFormDesigner.ClearSelection 174	ed_Designer.TzCustomFormDesigner.GridStepX 190
ed_Designer.TzCustomFormDesigner.ClearUndo 171	ed_Designer.TzCustomFormDesigner.GridStepY 190
ed_Designer.TzCustomFormDesigner.CloseDisactive 188	

---

---

ed_Designer.TzCustomFormDesigner.Groups 190	ed_Designer.TzCustomFormDesigner.OnGetComponentHint 198
ed_Designer.TzCustomFormDesigner.GuidelinesStyle 191	ed_Designer.TzCustomFormDesigner.OnGetComponentLocked 195
ed_Designer.TzCustomFormDesigner.IgnoreReadErrors 191	ed_Designer.TzCustomFormDesigner.OnGetMethodNames 198
ed_Designer.TzCustomFormDesigner.Intf_Notification 179	ed_Designer.TzCustomFormDesigner.OnGetObjectName 195
ed_Designer.TzCustomFormDesigner.IsComponentHidden 179	ed_Designer.TzCustomFormDesigner.OnGetScriptProc 199
ed_Designer.TzCustomFormDesigner.IsDesignMsg 180	ed_Designer.TzCustomFormDesigner.OnNotification 199
ed_Designer.TzCustomFormDesigner.IsLocked 179	ed_Designer.TzCustomFormDesigner.OnPopUndo 196
ed_Designer.TzCustomFormDesigner.IsProtected 180	ed_Designer.TzCustomFormDesigner.OnPushUndo 197
ed_Designer.TzCustomFormDesigner.IsRootSelected 180	ed_Designer.TzCustomFormDesigner.OnRenameMethod 200
ed_Designer.TzCustomFormDesigner.IsSourceReadOnly 180	ed_Designer.TzCustomFormDesigner.OnSetNewName 197
ed_Designer.TzCustomFormDesigner.KeyDown 180	ed_Designer.TzCustomFormDesigner.OnSetScriptProc 200
ed_Designer.TzCustomFormDesigner.KeyPress 181	ed_Designer.TzCustomFormDesigner.OnShowMethod 201
ed_Designer.TzCustomFormDesigner.KeyUp 181	ed_Designer.TzCustomFormDesigner.OnUpdateAction 198
ed_Designer.TzCustomFormDesigner.LoadFromFile 177	ed_Designer.TzCustomFormDesigner.OnValidateMethod 201
ed_Designer.TzCustomFormDesigner.LoadFromStream 178	ed_Designer.TzCustomFormDesigner.PaintControl 184
ed_Designer.TzCustomFormDesigner.LockControls 190	ed_Designer.TzCustomFormDesigner.PaintGrid 184
ed_Designer.TzCustomFormDesigner.LockPublished 190	ed_Designer.TzCustomFormDesigner.PasteSelection 184
ed_Designer.TzCustomFormDesigner.MethodExists 181	ed_Designer.TzCustomFormDesigner.PopupMenu 191
ed_Designer.TzCustomFormDesigner.Modified 181	ed_Designer.TzCustomFormDesigner.PopupMenuFilter 192
ed_Designer.TzCustomFormDesigner.MouseDown 182	ed_Designer.TzCustomFormDesigner.ReadComp 184
ed_Designer.TzCustomFormDesigner.MouseMove 182	ed_Designer.TzCustomFormDesigner.ReadOnly 191
ed_Designer.TzCustomFormDesigner.MouseUp 182	ed_Designer.TzCustomFormDesigner.Redo 179
ed_Designer.TzCustomFormDesigner.MultiSelect 190	ed_Designer.TzCustomFormDesigner.RenameMethod 184
ed_Designer.TzCustomFormDesigner.Navigate 182	ed_Designer.TzCustomFormDesigner.Root 192
ed_Designer.TzCustomFormDesigner.NoSelection 183	ed_Designer.TzCustomFormDesigner.RootModified 192
ed_Designer.TzCustomFormDesigner.Notification 183	ed_Designer.TzCustomFormDesigner.SaveToFile 179
ed_Designer.TzCustomFormDesigner.NotifySelChanged 183	ed_Designer.TzCustomFormDesigner.SaveToStream 180
ed_Designer.TzCustomFormDesigner.OnCanDelete 195	ed_Designer.TzCustomFormDesigner.Scale 185
ed_Designer.TzCustomFormDesigner.OnCanEdit 194	ed_Designer.TzCustomFormDesigner.SelCount 193
ed_Designer.TzCustomFormDesigner.OnCanInsert 195	ed_Designer.TzCustomFormDesigner.SelectAll 185
ed_Designer.TzCustomFormDesigner.OnCanMove 196	ed_Designer.TzCustomFormDesigner.SelectComponent 185
ed_Designer.TzCustomFormDesigner.OnCanRename 196	ed_Designer.TzCustomFormDesigner.Selected 193
ed_Designer.TzCustomFormDesigner.OnCanResize 196	ed_Designer.TzCustomFormDesigner.SelectedComponent 185
ed_Designer.TzCustomFormDesigner.OnCanSelect 196	ed_Designer.TzCustomFormDesigner.SelectedComponentsCount 180
ed_Designer.TzCustomFormDesigner.OnCreateComponent 197	ed_Designer.TzCustomFormDesigner.SelectionChanged 185
ed_Designer.TzCustomFormDesigner.OnCreateFrame 197	ed_Designer.TzCustomFormDesigner.SelectObj 185
ed_Designer.TzCustomFormDesigner.OnCreateIcon 197	ed_Designer.TzCustomFormDesigner.SelectRect 185
ed_Designer.TzCustomFormDesigner.OnCreateMethod 198	ed_Designer.TzCustomFormDesigner.SelMarker 193
ed_Designer.TzCustomFormDesigner.OnDrawControl 195	
ed_Designer.TzCustomFormDesigner.OnExecuteAction 195	
ed_Designer.TzCustomFormDesigner.OnFormClosed 198	

---

---

ed_Designer.TzCustomFormDesigner.SendToBack 186	ed_Designer.TzFormDesigner.OnCanDelete 217
ed_Designer.TzCustomFormDesigner.SetPasteName 186	ed_Designer.TzFormDesigner.OnCanEdit 217
ed_Designer.TzCustomFormDesigner.SetScriptEvent 186	ed_Designer.TzFormDesigner.OnCanInsert 217
ed_Designer.TzCustomFormDesigner.SetSelections 186	ed_Designer.TzFormDesigner.OnCanMove 217
ed_Designer.TzCustomFormDesigner.ShowCaptions 193	ed_Designer.TzFormDesigner.OnCanRename 218
ed_Designer.TzCustomFormDesigner.ShowMethod 186	ed_Designer.TzFormDesigner.OnCanResize 218
ed_Designer.TzCustomFormDesigner.ShowPopupMenu 186	ed_Designer.TzFormDesigner.OnCanSelect 218
ed_Designer.TzCustomFormDesigner.ShowTabOrder 182	ed_Designer.TzFormDesigner.OnCreateComponent 218
ed_Designer.TzCustomFormDesigner.SizeSelected 186	ed_Designer.TzFormDesigner.OnCreateFrame 219
ed_Designer.TzCustomFormDesigner.SnapToGrid 193	ed_Designer.TzFormDesigner.OnCreateIcon 219
ed_Designer.TzCustomFormDesigner.StartDrag 187	ed_Designer.TzFormDesigner.OnCreateMethod 219
ed_Designer.TzCustomFormDesigner.StoreEvents 191	ed_Designer.TzFormDesigner.OnDragDrop 219
ed_Designer.TzCustomFormDesigner.TabOrderIcons 191	ed_Designer.TzFormDesigner.OnDragOver 220
ed_Designer.TzCustomFormDesigner.Target 193	ed_Designer.TzFormDesigner.OnDrawControl 220
ed_Designer.TzCustomFormDesigner.TextEditMode 191	ed_Designer.TzFormDesigner.OnExecuteAction 220
ed_Designer.TzCustomFormDesigner.Undo 183	ed_Designer.TzFormDesigner.OnFormClosed 220
ed_Designer.TzCustomFormDesigner.UndoLimit 192	ed_Designer.TzFormDesigner.OnGetComponentHint 220
ed_Designer.TzCustomFormDesigner.UndoLoad 192	ed_Designer.TzFormDesigner.OnGetComponentLocked 221
ed_Designer.TzCustomFormDesigner.UniqueName 187	ed_Designer.TzFormDesigner.OnGetMethodNames 221
ed_Designer.TzCustomFormDesigner.UpdateAction 182	ed_Designer.TzFormDesigner.OnGetObjectName 221
ed_Designer.TzCustomFormDesigner.UpdateComplcons 187	ed_Designer.TzFormDesigner.OnGetScriptProc 222
ed_Designer.TzCustomFormDesigner.ValidateMethod 187	ed_Designer.TzFormDesigner.OnHandleControlMessage 222
ed_Designer.TzFormDesigner 201	ed_Designer.TzFormDesigner.OnKeyDown 222
ed_Designer.TzFormDesigner.Active 214	ed_Designer.TzFormDesigner.OnKeyPress 223
ed_Designer.TzFormDesigner.AllowComponents 214	ed_Designer.TzFormDesigner.OnKeyUp 223
ed_Designer.TzFormDesigner.AutoAlign 214	ed_Designer.TzFormDesigner.OnMouseDown 223
ed_Designer.TzFormDesigner.BDSStyle 214	ed_Designer.TzFormDesigner.OnMouseMove 223
ed_Designer.TzFormDesigner.CaptionFont 214	ed_Designer.TzFormDesigner.OnMouseUp 224
ed_Designer.TzFormDesigner.CloseDisactive 214	ed_Designer.TzFormDesigner.OnNotification 224
ed_Designer.TzFormDesigner.DesignSurface 215	ed_Designer.TzFormDesigner.OnPopUndo 224
ed_Designer.TzFormDesigner.DisplayControlGrid 215	ed_Designer.TzFormDesigner.OnPushUndo 224
ed_Designer.TzFormDesigner.DisplayGrid 215	ed_Designer.TzFormDesigner.OnRenameMethod 224
ed_Designer.TzFormDesigner.DragParentLimit 215	ed_Designer.TzFormDesigner.OnSetNewName 225
ed_Designer.TzFormDesigner.FlatIcons 215	ed_Designer.TzFormDesigner.OnSetScriptProc 225
ed_Designer.TzFormDesigner.GridStepX 215	ed_Designer.TzFormDesigner.OnShowMethod 225
ed_Designer.TzFormDesigner.GridStepY 216	ed_Designer.TzFormDesigner.OnUpdateAction 225
ed_Designer.TzFormDesigner.GuidelinesStyle 216	ed_Designer.TzFormDesigner.OnValidateMethod 225
ed_Designer.TzFormDesigner.IgnoreReadErrors 216	ed_Designer.TzFormDesigner.PopupMenu 226
ed_Designer.TzFormDesigner.LockControls 216	ed_Designer.TzFormDesigner.PopupMenuFilter 226
ed_Designer.TzFormDesigner.LockPublished 216	ed_Designer.TzFormDesigner.ReadOnly 226
ed_Designer.TzFormDesigner.MultiSelect 216	ed_Designer.TzFormDesigner.SelMarker 226
ed_Designer.TzFormDesigner.OnActiveChanged 217	ed_Designer.TzFormDesigner.ShowCaptions 227

---

- ed\_Designer.TzFormDesigner.ShowHints 227
- ed\_Designer.TzFormDesigner.SnapToGrid 227
- ed\_Designer.TzFormDesigner.StoreEvents 227
- ed\_Designer.TzFormDesigner.TabOrderIcons 227
- ed\_Designer.TzFormDesigner.Target 227
- ed\_Designer.TzFormDesigner.TextEditMode 228
- ed\_Designer.TzFormDesigner.UndoLimit 229
- ed\_DsnBase namespace
  - Classes 144
  - Types 156
- ed\_DsnBase.TBaseDesigner 144
- ed\_DsnBase.TBaseDesigner.Active 151
- ed\_DsnBase.TBaseDesigner.CanProcessNCMessages 146
- ed\_DsnBase.TBaseDesigner.Client2Screen 147
- ed\_DsnBase.TBaseDesigner.ClientOrg 147
- ed\_DsnBase.TBaseDesigner.Create 147
- ed\_DsnBase.TBaseDesigner.DesignState 147
- ed\_DsnBase.TBaseDesigner.Destroy 147
- ed\_DsnBase.TBaseDesigner.DoObjectHint 147
- ed\_DsnBase.TBaseDesigner.DragDrop 147
- ed\_DsnBase.TBaseDesigner.DragOver 148
- ed\_DsnBase.TBaseDesigner.HintObject 152
- ed\_DsnBase.TBaseDesigner.IsRTL 148
- ed\_DsnBase.TBaseDesigner.KeyDown 148
- ed\_DsnBase.TBaseDesigner.KeyPress 149
- ed\_DsnBase.TBaseDesigner.KeyUp 149
- ed\_DsnBase.TBaseDesigner.Loaded 149
- ed\_DsnBase.TBaseDesigner.MouseDown 150
- ed\_DsnBase.TBaseDesigner.MouseMove 150
- ed\_DsnBase.TBaseDesigner.MouseUp 150
- ed\_DsnBase.TBaseDesigner.OnActiveChanged 152
- ed\_DsnBase.TBaseDesigner.OnDragDrop 152
- ed\_DsnBase.TBaseDesigner.OnDragOver 152
- ed\_DsnBase.TBaseDesigner.OnHandleControlMessage 153
- ed\_DsnBase.TBaseDesigner.OnKeyDown 153
- ed\_DsnBase.TBaseDesigner.OnKeyPress 153
- ed\_DsnBase.TBaseDesigner.OnKeyUp 154
- ed\_DsnBase.TBaseDesigner.OnMouseDown 154
- ed\_DsnBase.TBaseDesigner.OnMouseMove 154
- ed\_DsnBase.TBaseDesigner.OnMouseUp 154
- ed\_DsnBase.TBaseDesigner.ProcessMessage 151
- ed\_DsnBase.TBaseDesigner.ResetHint 151
- ed\_DsnBase.TBaseDesigner.Screen2Client 151
- ed\_DsnBase.TBaseDesigner.SetActive 151
- ed\_DsnBase.TBaseDesigner.ShowHint 151
- ed\_DsnBase.TBaseDesigner.ShowHints 152
- ed\_DsnBase.TDesignOperation 155
- ed\_DsnBase.TDesignOperation enumeration 155
- ed\_DsnBase.TDesignOperations 156
- ed\_DsnBase.TDesignOperations type 156
- ed\_DsnBase.TDsnDragState 155
- ed\_DsnBase.TDsnDragState enumeration 155
- ed\_DsnBase.THandleControlMessage 156
- ed\_DsnBase.THandleControlMessage type 156
- ed\_dsncont namespace
  - Classes 237
  - Structs, Records, Enums 241
- ed\_dsncont.TDesignSurface 237
- ed\_dsncont.TDesignSurface.Activate 238
- ed\_dsncont.TDesignSurface.AdjustScroll 238
- ed\_dsncont.TDesignSurface.DoSizing 238
- ed\_dsncont.TDesignSurface.DsnShowFrame 239
- ed\_dsncont.TDesignSurface.ExecuteAction 238
- ed\_dsncont.TDesignSurface.FlatScrollBars 239
- ed\_dsncont.TDesignSurface.Form 240
- ed\_dsncont.TDesignSurface.FormOrigin 240
- ed\_dsncont.TDesignSurface.FrameSize 240
- ed\_dsncont.TDesignSurface.HideFormBorders 240
- ed\_dsncont.TDesignSurface.RulerClientArea 240
- ed\_dsncont.TDesignSurface.ScrollPos 240
- ed\_dsncont.TDesignSurface.ShowFrame 240
- ed\_dsncont.TDesignSurface.ShowRuler 241
- ed\_dsncont.TDesignSurface.UpdateAction 239
- ed\_dsncont.TDesignSurface.UseUnits 241
- ed\_dsncont.TRulerUnits 241
- ed\_dsncont.TRulerUnits enumeration 241
- ed\_ObjTree namespace
  - Classes 267
  - Functions 294
  - Types 294
- ed\_ObjTree.CreateGhostedImages 294
- ed\_ObjTree.CreateGhostedImages function 294
- ed\_ObjTree.TCreateSprigNodeEvent 295
- ed\_ObjTree.TCreateSprigNodeEvent type 295



---

ed_ObjTree.TCustomDesignerObjTree	267	ed_ObjTree.TDesignerObjTree.Indent	282
ed_ObjTree.TCustomDesignerObjTree.AddSprigAddItems	269	ed_ObjTree.TDesignerObjTree.Items	282
ed_ObjTree.TCustomDesignerObjTree.AddType	269	ed_ObjTree.TDesignerObjTree.MultiSelect	282
ed_ObjTree.TCustomDesignerObjTree.AddTypeCount	269	ed_ObjTree.TDesignerObjTree.MultiSelectStyle	282
ed_ObjTree.TCustomDesignerObjTree.AddTypes	271	ed_ObjTree.TDesignerObjTree.OnAddition	283
ed_ObjTree.TCustomDesignerObjTree.CanDelete	269	ed_ObjTree.TDesignerObjTree.OnAdvancedCustomDraw	283
ed_ObjTree.TCustomDesignerObjTree.CanMove	269	ed_ObjTree.TDesignerObjTree.OnAdvancedCustomDrawItem	283
ed_ObjTree.TCustomDesignerObjTree.Create	269	ed_ObjTree.TDesignerObjTree.OnChange	283
ed_ObjTree.TCustomDesignerObjTree.DeleteSelected	270	ed_ObjTree.TDesignerObjTree.OnChanging	284
ed_ObjTree.TCustomDesignerObjTree.Designer	271	ed_ObjTree.TDesignerObjTree.OnClick	284
ed_ObjTree.TCustomDesignerObjTree.Destroy	270	ed_ObjTree.TDesignerObjTree.OnCollapsed	284
ed_ObjTree.TCustomDesignerObjTree.Loaded	270	ed_ObjTree.TDesignerObjTree.OnCollapsing	284
ed_ObjTree.TCustomDesignerObjTree.Move	270	ed_ObjTree.TDesignerObjTree.OnCompare	284
ed_ObjTree.TCustomDesignerObjTree.Notification	270	ed_ObjTree.TDesignerObjTree.OnContextPopup	285
ed_ObjTree.TCustomDesignerObjTree.OnCreateSprigNode	271	ed_ObjTree.TDesignerObjTree.OnCreateNodeClass	285
ed_ObjTree.TCustomDesignerObjTree.RootSprig	271	ed_ObjTree.TDesignerObjTree.OnCreateSprigNode	285
ed_ObjTree.TDesignerObjTree	271	ed_ObjTree.TDesignerObjTree.OnCustomDraw	285
ed_ObjTree.TDesignerObjTree.Align	277	ed_ObjTree.TDesignerObjTree.OnCustomDrawItem	286
ed_ObjTree.TDesignerObjTree.Anchors	277	ed_ObjTree.TDesignerObjTree.OnDbClick	286
ed_ObjTree.TDesignerObjTree.AutoExpand	278	ed_ObjTree.TDesignerObjTree.OnDeletion	286
ed_ObjTree.TDesignerObjTree.BevelEdges	278	ed_ObjTree.TDesignerObjTree.OnDragDrop	286
ed_ObjTree.TDesignerObjTree.BevelInner	278	ed_ObjTree.TDesignerObjTree.OnDragOver	286
ed_ObjTree.TDesignerObjTree.BevelKind	278	ed_ObjTree.TDesignerObjTree.OnEdited	287
ed_ObjTree.TDesignerObjTree.BevelOuter	279	ed_ObjTree.TDesignerObjTree.OnEditing	287
ed_ObjTree.TDesignerObjTree.BevelWidth	279	ed_ObjTree.TDesignerObjTree.OnEndDock	287
ed_ObjTree.TDesignerObjTree.BiDiMode	279	ed_ObjTree.TDesignerObjTree.OnEndDrag	287
ed_ObjTree.TDesignerObjTree.BorderStyle	279	ed_ObjTree.TDesignerObjTree.OnEnter	288
ed_ObjTree.TDesignerObjTree.BorderWidth	279	ed_ObjTree.TDesignerObjTree.OnExit	288
ed_ObjTree.TDesignerObjTree.ChangeDelay	280	ed_ObjTree.TDesignerObjTree.OnExpanded	288
ed_ObjTree.TDesignerObjTree.Color	280	ed_ObjTree.TDesignerObjTree.OnExpanding	288
ed_ObjTree.TDesignerObjTree.Constraints	280	ed_ObjTree.TDesignerObjTree.OnGetImageIndex	288
ed_ObjTree.TDesignerObjTree.Create	277	ed_ObjTree.TDesignerObjTree.OnGetSelectedIndex	288
ed_ObjTree.TDesignerObjTree.Ctl3D	280	ed_ObjTree.TDesignerObjTree.OnKeyDown	289
ed_ObjTree.TDesignerObjTree.DragCursor	280	ed_ObjTree.TDesignerObjTree.OnKeyPress	289
ed_ObjTree.TDesignerObjTree.DragKind	281	ed_ObjTree.TDesignerObjTree.OnKeyUp	289
ed_ObjTree.TDesignerObjTree.DragMode	281	ed_ObjTree.TDesignerObjTree.OnMouseDown	289
ed_ObjTree.TDesignerObjTree.Enabled	281	ed_ObjTree.TDesignerObjTree.OnMouseMove	290
ed_ObjTree.TDesignerObjTree.Font	281	ed_ObjTree.TDesignerObjTree.OnMouseUp	290
ed_ObjTree.TDesignerObjTree.HideSelection	281	ed_ObjTree.TDesignerObjTree.OnStartDock	290
ed_ObjTree.TDesignerObjTree.HotTrack	281	ed_ObjTree.TDesignerObjTree.OnStartDrag	291
ed_ObjTree.TDesignerObjTree.Images	282	ed_ObjTree.TDesignerObjTree.ParentBiDiMode	291

---

- ed\_ObjTree.TDesignerObjTree.ParentColor 291
- ed\_ObjTree.TDesignerObjTree.ParentCtl3D 291
- ed\_ObjTree.TDesignerObjTree.ParentFont 291
- ed\_ObjTree.TDesignerObjTree.ParentShowHint 291
- ed\_ObjTree.TDesignerObjTree.PopupMenu 292
- ed\_ObjTree.TDesignerObjTree.ReadOnly 292
- ed\_ObjTree.TDesignerObjTree.RightClickSelect 292
- ed\_ObjTree.TDesignerObjTree.RowSelect 292
- ed\_ObjTree.TDesignerObjTree.ShowButtons 292
- ed\_ObjTree.TDesignerObjTree.ShowHint 293
- ed\_ObjTree.TDesignerObjTree.ShowLines 293
- ed\_ObjTree.TDesignerObjTree.ShowRoot 293
- ed\_ObjTree.TDesignerObjTree.SortType 293
- ed\_ObjTree.TDesignerObjTree.StateImages 293
- ed\_ObjTree.TDesignerObjTree.TabOrder 293
- ed\_ObjTree.TDesignerObjTree.TabStop 293
- ed\_ObjTree.TDesignerObjTree.ToolTips 294
- ed\_ObjTree.TDesignerObjTree.Visible 294
- ed\_RegComps namespace
  - Classes 242
  - Functions 256
  - Structs, Records, Enums 257
  - Types 258
  - Variables 258
- ed\_RegComps.DrawBtnIcon 257
- ed\_RegComps.DrawBtnIcon function 257
- ed\_RegComps.Frames 242
- ed\_RegComps.PackageMng 258
- ed\_RegComps.PackageMng variable 258
- ed\_RegComps.Runtime 259
- ed\_RegComps.Runtime variable 259
- ed\_RegComps.TComponentClassInfo 242
- ed\_RegComps.TComponentClassInfo.ComponentClass 244
- ed\_RegComps.TComponentClassInfo.Create 243
- ed\_RegComps.TComponentClassInfo.Destroy 243
- ed\_RegComps.TComponentClassInfo.DisplayName 244
- ed\_RegComps.TComponentClassInfo.Hidden 244
- ed\_RegComps.TComponentClassInfo.Icon 244
- ed\_RegComps.TComponentClassInfo.InitPage 244
- ed\_RegComps.TComponentClassInfo.IsIconValid 244
- ed\_RegComps.TComponentClassInfo.Module 244
- ed\_RegComps.TComponentClassInfo.Page 245
- ed\_RegComps.TComponentRegEvent 258
- ed\_RegComps.TComponentRegEvent type 258
- ed\_RegComps.TComponentRegInfoEvent 258
- ed\_RegComps.TComponentRegInfoEvent type 258
- ed\_RegComps.TCustomModuleInfo 245
- ed\_RegComps.TFrameInfo 245
- ed\_RegComps.TFrameInfo.FrameClass 246
- ed\_RegComps.TFrameInfo.FrameResource 246
- ed\_RegComps.TIconBtnStyle 257
- ed\_RegComps.TIconBtnStyle enumeration 257
- ed\_RegComps.TPackageInfo 246
- ed\_RegComps.TPackageInfo.Active 247
- ed\_RegComps.TPackageInfo.Create 247
- ed\_RegComps.TPackageInfo.Description 247
- ed\_RegComps.TPackageInfo.Destroy 247
- ed\_RegComps.TPackageInfo.FileName 247
- ed\_RegComps.TPackageInfo.Handle 247
- ed\_RegComps.TPackageInfo.Requires 247
- ed\_RegComps.TPackageInfo.Units 248
- ed\_RegComps.TPackageMng 248
- ed\_RegComps.TPackageMng.AddComponent 250
- ed\_RegComps.TPackageMng.AddPackage 250
- ed\_RegComps.TPackageMng.AutoSave 255
- ed\_RegComps.TPackageMng.BeginUpdate 250
- ed\_RegComps.TPackageMng.ComponentCount 255
- ed\_RegComps.TPackageMng.Components 255
- ed\_RegComps.TPackageMng.Create 250
- ed\_RegComps.TPackageMng.CreateFrame 251
- ed\_RegComps.TPackageMng.CustomizePackages 251
- ed\_RegComps.TPackageMng.CustomizePalette 251
- ed\_RegComps.TPackageMng.DeleteComponent 251
- ed\_RegComps.TPackageMng.Destroy 251
- ed\_RegComps.TPackageMng.EndUpdate 252
- ed\_RegComps.TPackageMng.FindClass 252
- ed\_RegComps.TPackageMng.FindClassName 252
- ed\_RegComps.TPackageMng.FindClassNameIdx 252
- ed\_RegComps.TPackageMng.FindPackage 252
- ed\_RegComps.TPackageMng.FrameInfos 255
- ed\_RegComps.TPackageMng.GetCustomModule 252
- ed\_RegComps.TPackageMng.IsNolcon 253
- ed\_RegComps.TPackageMng.LoadPaletaFromIni 253
- ed\_RegComps.TPackageMng.OnRegisterComponent 256

- ed\_RegComps.TPackageMng.OnRegisterComponentInfo 256
- ed\_RegComps.TPackageMng.OnUnRegisterComponentInfo 256
- ed\_RegComps.TPackageMng.Packages 255
- ed\_RegComps.TPackageMng.Pages 256
- ed\_RegComps.TPackageMng.ReadRegInfo 253
- ed\_RegComps.TPackageMng.RegisterFrame 253
- ed\_RegComps.TPackageMng.RegSubkey 256
- ed\_RegComps.TPackageMng.RemoveEmptyPages 253
- ed\_RegComps.TPackageMng.RemovePackage 254
- ed\_RegComps.TPackageMng.RenamePage 254
- ed\_RegComps.TPackageMng.ResetPalette 254
- ed\_RegComps.TPackageMng.SavePaletteToIni 254
- ed\_RegComps.TPackageMng.SaveRegInfo 254
- ed\_RegComps.TPackageMng.SetComponentOrder 255
- ed\_RegMeth namespace
  - Classes 259
  - Structs, Records, Enums 265
  - Types 266
  - Variables 266
- ed\_RegMeth.MethRegister 267
- ed\_RegMeth.MethRegister variable 267
- ed\_RegMeth.PMethod 266
- ed\_RegMeth.PMethod type 266
- ed\_RegMeth.PMethodInfo 266
- ed\_RegMeth.PMethodInfo type 266
- ed\_RegMeth.TDefaultMethodRegister 259
- ed\_RegMeth.TDefaultMethodRegister.Add 261, 262, 263, 264
- ed\_RegMeth.TDefaultMethodRegister.AddMethod 264
- ed\_RegMeth.TDefaultMethodRegister.Count 265
- ed\_RegMeth.TDefaultMethodRegister.Create 264
- ed\_RegMeth.TDefaultMethodRegister.Destroy 264
- ed\_RegMeth.TDefaultMethodRegister.FInfos 261
- ed\_RegMeth.TDefaultMethodRegister.GetMethodsNames 264
- ed\_RegMeth.TDefaultMethodRegister.Items 265
- ed\_RegMeth.TDefaultMethodRegister.RemoveObject 265
- ed\_RegMeth.TDefaultMethodRegister.ValidateMethod 265
- ed\_RegMeth.TMethodInfo 265
- ed\_RegMeth.TMethodInfo record 265
- ed\_TextEdit namespace
  - Classes 295
  - Functions 306
  - Types 307
- ed\_TextEdit.CreateImplEditor 306
- ed\_TextEdit.CreateImplEditor function 306
- ed\_TextEdit.RegisterInplaceComponentEditor 306
- ed\_TextEdit.RegisterInplaceComponentEditor function 306
- ed\_TextEdit.TDsnInplaceEditor 295
- ed\_TextEdit.TDsnInplaceEditor.Adapter 300
- ed\_TextEdit.TDsnInplaceEditor.Alignment 300
- ed\_TextEdit.TDsnInplaceEditor.Close 299
- ed\_TextEdit.TDsnInplaceEditor.Color 300
- ed\_TextEdit.TDsnInplaceEditor.Create 299
- ed\_TextEdit.TDsnInplaceEditor.Destroy 300
- ed\_TextEdit.TDsnInplaceEditor.IsUnicode 301
- ed\_TextEdit.TDsnInplaceEditor.LoadText 300
- ed\_TextEdit.TDsnInplaceEditor.MultiLine 301
- ed\_TextEdit.TDsnInplaceEditor.OnChange 301
- ed\_TextEdit.TDsnInplaceEditor.OnExit 301
- ed\_TextEdit.TDsnInplaceEditor.SaveText 300
- ed\_TextEdit.TDsnInplaceEditor.TextW 301
- ed\_TextEdit.TDsnInplaceEditor.WordWrap 302
- ed\_TextEdit.TInplaceComponentEditor 302
- ed\_TextEdit.TInplaceComponentEditor.BoundsRect 305
- ed\_TextEdit.TInplaceComponentEditor.Control 305
- ed\_TextEdit.TInplaceComponentEditor.Create 303
- ed\_TextEdit.TInplaceComponentEditor.GetBoundsRect 303
- ed\_TextEdit.TInplaceComponentEditor.GetText 304
- ed\_TextEdit.TInplaceComponentEditor.GetTextW 304
- ed\_TextEdit.TInplaceComponentEditor.HandlePos 304
- ed\_TextEdit.TInplaceComponentEditor.IsAutoUpdate 304
- ed\_TextEdit.TInplaceComponentEditor.IsUnicode 304
- ed\_TextEdit.TInplaceComponentEditor.SetEditor 304
- ed\_TextEdit.TInplaceComponentEditor.SetHitPoint 305
- ed\_TextEdit.TInplaceComponentEditor.SetText 305
- ed\_TextEdit.TInplaceComponentEditor.SetTextW 305
- ed\_TextEdit.TInplaceComponentEditor.Text 305
- ed\_TextEdit.TInplaceComponentEditor.TextW 306
- ed\_TextEdit.TInplaceComponentEditorClass 307
- ed\_TextEdit.TInplaceComponentEditorClass type 307
- edActns namespace
  - Classes 307
- edActns.TDesignerAction 308
- edActns.TDesignerAction.Caption 309
- edActns.TDesignerAction.Enabled 309

edActns.TDesignerAction.HelpContext 309	edActns.TdsnSelectAll 330
edActns.TDesignerAction.HelpKeyword 310	edActns.TdsnSendToBack 330
edActns.TDesignerAction.HelpType 310	edActns.TdsnSendToBack.Execute 332
edActns.TDesignerAction.Hint 310	edActns.TdsnShowTabOrder 332
edActns.TDesignerAction.ImageIndex 310	edActns.TdsnShowTabOrder.Execute 333
edActns.TDesignerAction.OnExecute 310	edActns.TdsnShowTabOrder.Update 333
edActns.TDesignerAction.OnHint 310	edActns.TdsnSizeDlg 334
edActns.TDesignerAction.OnUpdate 311	edActns.TdsnSizeDlg.Execute 335
edActns.TDesignerAction.SecondaryShortCuts 311	edActns.TdsnTabOrderDlg 335
edActns.TDesignerAction.ShortCut 311	edActns.TdsnTabOrderDlg.Execute 336
edActns.TDesignerAction.Update 309	edActns.TdsnTargetAction 337
edActns.TDesignerAction.Visible 311	edActns.TdsnTextEditMode 337
edActns.TdsnAlignmentDlg 311	edActns.TdsnTextEditMode.Execute 338
edActns.TdsnAlignmentDlg.Execute 313	edActns.TdsnTextEditMode.Update 338
edActns.TdsnAlignToGrid 313	edActns.TdsnUndo 338
edActns.TdsnAlignToGrid.Execute 314	edActns.TdsnUngroupControls 339
edActns.TdsnBringToFront 315	edActns.TdsnUngroupControls.Execute 340
edActns.TdsnBringToFront.Execute 316	edcCmbCombo namespace
edActns.TdsnCopy 316	Classes 340
edActns.TdsnCreationOrderDlg 317	Types 358
edActns.TdsnCreationOrderDlg.Execute 318	edcCmbCombo.TCanAddObjectEvent 359
edActns.TdsnCut 318	edcCmbCombo.TCanAddObjectEvent type 359
edActns.TdsnDelete 318	edcCmbCombo.TComponentCombo 341
edActns.TdsnDesignMode 319	edcCmbCombo.TComponentCombo.Align 346
edActns.TdsnDesignMode.Execute 320	edcCmbCombo.TComponentCombo.Anchors 346
edActns.TdsnDesignMode.Update 320	edcCmbCombo.TComponentCombo.AutoCloseUp 347
edActns.TdsnFlipChildren 320	edcCmbCombo.TComponentCombo.AutoDropDown 347
edActns.TdsnFlipChildren.Execute 322	edcCmbCombo.TComponentCombo.AutoHint 347
edActns.TdsnFlipChildrenAll 322	edcCmbCombo.TComponentCombo.BiDiMode 347
edActns.TdsnFlipChildrenAll.Execute 323	edcCmbCombo.TComponentCombo.Change 344
edActns.TdsnGroupControls 323	edcCmbCombo.TComponentCombo.ClassNameColor 347
edActns.TdsnGroupControls.Execute 325	edcCmbCombo.TComponentCombo.ClassNameDelim 348
edActns.TdsnGroupControls.Update 325	edcCmbCombo.TComponentCombo.Color 348
edActns.TdsnLockControls 325	edcCmbCombo.TComponentCombo.Constraints 348
edActns.TdsnLockControls.Execute 326	edcCmbCombo.TComponentCombo.Create 344
edActns.TdsnLockControls.Update 326	edcCmbCombo.TComponentCombo.Ctl3D 348
edActns.TdsnPaste 327	edcCmbCombo.TComponentCombo.Designer 348
edActns.TdsnRedo 327	edcCmbCombo.TComponentCombo.Destroy 345
edActns.TdsnScale 327	edcCmbCombo.TComponentCombo.DoAddObject 345
edActns.TdsnScale.Execute 328	edcCmbCombo.TComponentCombo.DragCursor 348
edActns.TdsnSelAction 329	edcCmbCombo.TComponentCombo.DragKind 349
edActns.TdsnSelAction.Update 330	edcCmbCombo.TComponentCombo.DragMode 349

edcCmbCombo.TComponentCombo.DropDownCount	349	edcCmbCombo.TComponentCombo.ShowClassName	356
edcCmbCombo.TComponentCombo.DropDownWidth	349	edcCmbCombo.TComponentCombo.ShowComponents	357
edcCmbCombo.TComponentCombo.Enabled	349	edcCmbCombo.TComponentCombo.ShowHint	357
edcCmbCombo.TComponentCombo.FillObjList	345	edcCmbCombo.TComponentCombo.Sorted	357
edcCmbCombo.TComponentCombo.Font	350	edcCmbCombo.TComponentCombo.TabOrder	357
edcCmbCombo.TComponentCombo.ImeMode	350	edcCmbCombo.TComponentCombo.TabStop	357
edcCmbCombo.TComponentCombo.ImeName	350	edcCmbCombo.TComponentCombo.Text	357
edcCmbCombo.TComponentCombo.IncludeContainer	350	edcCmbCombo.TComponentCombo.UpdateObjectList	345
edcCmbCombo.TComponentCombo.ItemHeight	350	edcCmbCombo.TComponentCombo.Visible	358
edcCmbCombo.TComponentCombo.MaxLength	351	edcCmbCombo.TGetClassNameEvent	359
edcCmbCombo.TComponentCombo.NameColor	351	edcCmbCombo.TGetClassNameEvent type	359
edcCmbCombo.TComponentCombo.Notification	345	edcCmbCombo.TGetComponentsEvent	359
edcCmbCombo.TComponentCombo.OnCanAddObject	358	edcCmbCombo.TGetComponentsEvent type	359
edcCmbCombo.TComponentCombo.OnClick	351	edcCmbCombo.TSelChangedEvent	359
edcCmbCombo.TComponentCombo.OnCloseUp	351	edcCmbCombo.TSelChangedEvent type	359
edcCmbCombo.TComponentCombo.OnContextPopup	351	edcCompPal namespace	
edcCmbCombo.TComponentCombo.OnDblClick	352	Classes	360
edcCmbCombo.TComponentCombo.OnDragDrop	352	edcCompPal.TPalettePanel	360
edcCmbCombo.TComponentCombo.OnDragOver	352	edcCompPal.TPalettePanel.Align	366
edcCmbCombo.TComponentCombo.OnDrawItem	353	edcCompPal.TPalettePanel.Anchors	366
edcCmbCombo.TComponentCombo.OnDropDown	353	edcCompPal.TPalettePanel.AutoSize	367
edcCmbCombo.TComponentCombo.OnEndDock	353	edcCompPal.TPalettePanel.BevelInner	367
edcCmbCombo.TComponentCombo.OnEndDrag	353	edcCompPal.TPalettePanel.BevelOuter	367
edcCmbCombo.TComponentCombo.OnEnter	353	edcCompPal.TPalettePanel.BevelWidth	367
edcCmbCombo.TComponentCombo.OnExit	354	edcCompPal.TPalettePanel.BiDiMode	368
edcCmbCombo.TComponentCombo.OnGetClassName	358	edcCompPal.TPalettePanel.BorderStyle	368
edcCmbCombo.TComponentCombo.OnGetComponents	358	edcCompPal.TPalettePanel.BorderWidth	368
edcCmbCombo.TComponentCombo.OnKeyDown	354	edcCompPal.TPalettePanel.ButtonClick	364
edcCmbCombo.TComponentCombo.OnKeyPress	354	edcCompPal.TPalettePanel.ButtonHeight	368
edcCmbCombo.TComponentCombo.OnKeyUp	354	edcCompPal.TPalettePanel.ButtonWidth	369
edcCmbCombo.TComponentCombo.OnMeasureItem	355	edcCompPal.TPalettePanel.Color	369
edcCmbCombo.TComponentCombo.OnSelChanged	358	edcCompPal.TPalettePanel.Constraints	369
edcCmbCombo.TComponentCombo.OnSelect	355	edcCompPal.TPalettePanel.Create	365
edcCmbCombo.TComponentCombo.OnStartDock	355	edcCompPal.TPalettePanel.Ctl3D	369
edcCmbCombo.TComponentCombo.OnStartDrag	355	edcCompPal.TPalettePanel.Destroy	365
edcCmbCombo.TComponentCombo.ParentBiDiMode	356	edcCompPal.TPalettePanel.DownButton	370
edcCmbCombo.TComponentCombo.ParentColor	356	edcCompPal.TPalettePanel.DragCursor	370
edcCmbCombo.TComponentCombo.ParentCtl3D	356	edcCompPal.TPalettePanel.DragImageType	370
edcCmbCombo.TComponentCombo.ParentFont	356	edcCompPal.TPalettePanel.DragKind	370
edcCmbCombo.TComponentCombo.ParentShowHint	356	edcCompPal.TPalettePanel.DragMode	370
edcCmbCombo.TComponentCombo.PopupMenu	356	edcCompPal.TPalettePanel.DrawButton	365
edcCmbCombo.TComponentCombo.SetSelection	345	edcCompPal.TPalettePanel.Enabled	370

edcCompPal.TPalettePanel.Flat 371	edcCompPal.TPaletteTab.DragKind 381
edcCompPal.TPalettePanel.Font 371	edcCompPal.TPaletteTab.DragMode 381
edcCompPal.TPalettePanel.GetButtonHint 365	edcCompPal.TPaletteTab.Enabled 381
edcCompPal.TPalettePanel.HintProps 371	edcCompPal.TPaletteTab.Flat 382
edcCompPal.TPalettePanel.Margins 371	edcCompPal.TPaletteTab.Font 382
edcCompPal.TPalettePanel.OnButtonClick 371	edcCompPal.TPaletteTab.HintProps 382
edcCompPal.TPalettePanel.OnCanResize 372	edcCompPal.TPaletteTab.HotTrack 382
edcCompPal.TPalettePanel.OnClick 372	edcCompPal.TPaletteTab.Images 382
edcCompPal.TPalettePanel.OnConstrainedResize 372	edcCompPal.TPaletteTab.MultiLine 382
edcCompPal.TPalettePanel.OnContextPopup 372	edcCompPal.TPaletteTab.OnChange 383
edcCompPal.TPalettePanel.OnDbClick 373	edcCompPal.TPaletteTab.OnChanging 383
edcCompPal.TPalettePanel.OnDragDrop 373	edcCompPal.TPaletteTab.OnContextPopup 383
edcCompPal.TPalettePanel.OnDragOver 373	edcCompPal.TPaletteTab.OnDragDrop 383
edcCompPal.TPalettePanel.OnEndDrag 374	edcCompPal.TPaletteTab.OnDragOver 383
edcCompPal.TPalettePanel.OnEnter 374	edcCompPal.TPaletteTab.OnDrawTab 383
edcCompPal.TPalettePanel.OnExit 374	edcCompPal.TPaletteTab.OnEndDock 384
edcCompPal.TPalettePanel.OnMouseDown 374	edcCompPal.TPaletteTab.OnEndDrag 384
edcCompPal.TPalettePanel.OnMouseMove 374	edcCompPal.TPaletteTab.OnEnter 384
edcCompPal.TPalettePanel.OnMouseUp 375	edcCompPal.TPaletteTab.OnExit 384
edcCompPal.TPalettePanel.OnResize 375	edcCompPal.TPaletteTab.OnGetImageIndex 384
edcCompPal.TPalettePanel.OnStartDrag 375	edcCompPal.TPaletteTab.OnMouseDown 384
edcCompPal.TPalettePanel.Orientation 375	edcCompPal.TPaletteTab.OnMouseMove 385
edcCompPal.TPalettePanel.Page 376	edcCompPal.TPaletteTab.OnMouseUp 385
edcCompPal.TPalettePanel.ParentBiDiMode 376	edcCompPal.TPaletteTab.OnResize 385
edcCompPal.TPalettePanel.ParentColor 376	edcCompPal.TPaletteTab.OnStartDrag 385
edcCompPal.TPalettePanel.ParentCtl3D 376	edcCompPal.TPaletteTab.OwnerDraw 385
edcCompPal.TPalettePanel.ParentFont 376	edcCompPal.TPaletteTab.PalettePanel 386
edcCompPal.TPalettePanel.ParentShowHint 376	edcCompPal.TPaletteTab.ParentBiDiMode 386
edcCompPal.TPalettePanel.PopupMenu 376	edcCompPal.TPaletteTab.ParentFont 386
edcCompPal.TPalettePanel.RowCount 376	edcCompPal.TPaletteTab.ParentShowHint 386
edcCompPal.TPalettePanel.ShowHint 377	edcCompPal.TPaletteTab.PopupMenu 386
edcCompPal.TPalettePanel.TabOrder 377	edcCompPal.TPaletteTab.RaggedRight 386
edcCompPal.TPalettePanel.TabStop 377	edcCompPal.TPaletteTab.ResetOnChange 386
edcCompPal.TPalettePanel.Transparent 377	edcCompPal.TPaletteTab.ScrollOpposite 387
edcCompPal.TPalettePanel.UpdateList 366	edcCompPal.TPaletteTab.ShowHint 387
edcCompPal.TPalettePanel.Visible 377	edcCompPal.TPaletteTab.Style 387
edcCompPal.TPaletteTab 377	edcCompPal.TPaletteTab.TabHeight 387
edcCompPal.TPaletteTab.Align 380	edcCompPal.TPaletteTab.TabIndex 387
edcCompPal.TPaletteTab.Anchors 380	edcCompPal.TPaletteTab.TabOrder 387
edcCompPal.TPaletteTab.BiDiMode 381	edcCompPal.TPaletteTab.TabPosition 388
edcCompPal.TPaletteTab.Constraints 381	edcCompPal.TPaletteTab.Tabs 388
edcCompPal.TPaletteTab.DragCursor 381	edcCompPal.TPaletteTab.TabStop 388

- edcCompPal.TPaletteTab.TabWidth 388
- edcCompPal.TPaletteTab.Visible 388
- edcDsnEvents namespace
  - Classes 389
  - Types 394
- edcDsnEvents.TDesignerEvent 394
- edcDsnEvents.TDesignerEvent type 394
- edcDsnEvents.TDesignerEvents 389
- edcDsnEvents.TDesignerEvents.OnActiveDsnChanged 391
- edcDsnEvents.TDesignerEvents.OnClassChanged 391
- edcDsnEvents.TDesignerEvents.OnDesignerClosed 391
- edcDsnEvents.TDesignerEvents.OnDesignerInitialized 391
- edcDsnEvents.TDesignerEvents.OnDsnKeyDown 392
- edcDsnEvents.TDesignerEvents.OnGetGlobalComponents 392
- edcDsnEvents.TDesignerEvents.OnGetWorkspaceOrigin 392
- edcDsnEvents.TDesignerEvents.OnItemDeleted 392
- edcDsnEvents.TDesignerEvents.OnItemInserted 393
- edcDsnEvents.TDesignerEvents.OnItemsModified 393
- edcDsnEvents.TDesignerEvents.OnKeyPress 393
- edcDsnEvents.TDesignerEvents.OnPaletteChanged 393
- edcDsnEvents.TDesignerEvents.OnRegisterComponent 394
- edcDsnEvents.TDesignerEvents.OnSelectionChanged 394
- edcDsnEvents.TDsnItemEvent 395
- edcDsnEvents.TDsnItemEvent type 395
- edcDsnEvents.TDsnKeyDownEvent 395
- edcDsnEvents.TDsnKeyDownEvent type 395
- edcDsnEvents.TDsnKeyPressEvent 395
- edcDsnEvents.TDsnKeyPressEvent type 395
- edcDsnEvents.TGetGlobalsEvent 395
- edcDsnEvents.TGetGlobalsEvent type 395
- edcDsnEvents.TOnGetPoint 395
- edcDsnEvents.TOnGetPoint type 395
- edcDsnEvents.TRegisterComponentEvent 396
- edcDsnEvents.TRegisterComponentEvent type 396
- edcPropCtrl namespace
  - Classes 396
  - Structs, Records, Enums 445
  - Types 446
- edcPropCtrl.IFormDesigner 446
- edcPropCtrl.IFormDesigner type 446
- edcPropCtrl.IProperty 446
- edcPropCtrl.IProperty type 446
- edcPropCtrl.TAcceptCategoryEvent 447
- edcPropCtrl.TAcceptCategoryEvent type 447
- edcPropCtrl.TAcceptPropertyEvent 447
- edcPropCtrl.TAcceptPropertyEvent type 447
- edcPropCtrl.TCategoryNode 396
- edcPropCtrl.TCategoryNode.Clear 398
- edcPropCtrl.TCategoryNode.Create 398
- edcPropCtrl.TCategoryNode.Expandable 398
- edcPropCtrl.TCategoryNode.GetName 399
- edcPropCtrl.TCategoryNode.HasValue 399
- edcPropCtrl.TChangeSelectionEvent 447
- edcPropCtrl.TChangeSelectionEvent type 447
- edcPropCtrl.TCustomInspectorList 399
- edcPropCtrl.TCustomInspectorList.AcceptProperty 404
- edcPropCtrl.TCustomInspectorList.ByCategories 407
- edcPropCtrl.TCustomInspectorList.Categories 408
- edcPropCtrl.TCustomInspectorList.cCategories 408
- edcPropCtrl.TCustomInspectorList.cDefValues 408
- edcPropCtrl.TCustomInspectorList.cEditBackGround 408
- edcPropCtrl.TCustomInspectorList.cEditValue 408
- edcPropCtrl.TCustomInspectorList.Component 408
- edcPropCtrl.TCustomInspectorList.CopyName 404
- edcPropCtrl.TCustomInspectorList.CopyValue 405
- edcPropCtrl.TCustomInspectorList.cPropName 409
- edcPropCtrl.TCustomInspectorList.cPropReadOnly 409
- edcPropCtrl.TCustomInspectorList.cPropReference 409
- edcPropCtrl.TCustomInspectorList.cPropValue 409
- edcPropCtrl.TCustomInspectorList.Create 405
- edcPropCtrl.TCustomInspectorList.CreateEditor 405
- edcPropCtrl.TCustomInspectorList.CreateItems 405
- edcPropCtrl.TCustomInspectorList.cSubProperty 409
- edcPropCtrl.TCustomInspectorList.CutValue 405
- edcPropCtrl.TCustomInspectorList.DefPropNameDraw 409
- edcPropCtrl.TCustomInspectorList.Designer 409
- edcPropCtrl.TCustomInspectorList.DoPrepareCanvas 405
- edcPropCtrl.TCustomInspectorList.DrawPropCell 406
- edcPropCtrl.TCustomInspectorList.EditedObject 410
- edcPropCtrl.TCustomInspectorList.ExpandRefs 410
- edcPropCtrl.TCustomInspectorList.FocusEditor 406
- edcPropCtrl.TCustomInspectorList.GetDesigner 406
- edcPropCtrl.TCustomInspectorList.HiddenCount 410
- edcPropCtrl.TCustomInspectorList.HintProps 410

edcPropCtrl.TCustomInspectorList.IncludeRefs 410  
edcPropCtrl.TCustomInspectorList.IsPropReadOnly 406  
edcPropCtrl.TCustomInspectorList.KeyDown 406  
edcPropCtrl.TCustomInspectorList.Loaded 406  
edcPropCtrl.TCustomInspectorList.MarkNonDefault 410  
edcPropCtrl.TCustomInspectorList.MouseDown 406  
edcPropCtrl.TCustomInspectorList.OnAcceptCategory 412  
edcPropCtrl.TCustomInspectorList.OnAcceptProperty 412  
edcPropCtrl.TCustomInspectorList.OnChangeSelection 412  
edcPropCtrl.TCustomInspectorList.OnGetPropReadOnly 413  
edcPropCtrl.TCustomInspectorList.OnPropListUpdated 413  
edcPropCtrl.TCustomInspectorList.OnPropValueChanged 413  
edcPropCtrl.TCustomInspectorList.OnSetPropValueA 413  
edcPropCtrl.TCustomInspectorList.OnSetPropValueW 413  
edcPropCtrl.TCustomInspectorList.PasteValue 406  
edcPropCtrl.TCustomInspectorList.PopupListAlign 411  
edcPropCtrl.TCustomInspectorList.PropValueChanged 407  
edcPropCtrl.TCustomInspectorList.ReadOnly 411  
edcPropCtrl.TCustomInspectorList.SaveValue 407  
edcPropCtrl.TCustomInspectorList.SearchPropKey 411  
edcPropCtrl.TCustomInspectorList.SearchPropMode 411  
edcPropCtrl.TCustomInspectorList.Selected 411  
edcPropCtrl.TCustomInspectorList.SelectedCount 411  
edcPropCtrl.TCustomInspectorList.SetItemIndex 407  
edcPropCtrl.TCustomInspectorList.ShowReadOnly 411  
edcPropCtrl.TCustomInspectorList.TypeKinds 411  
edcPropCtrl.TCustomInspectorList.TypeSelector 412  
edcPropCtrl.TCustomInspectorList.UpdateEditor 407  
edcPropCtrl.TCustomInspectorList.UpdateList 407  
edcPropCtrl.TGetPropReadOnlyEvent 447  
edcPropCtrl.TGetPropReadOnlyEvent type 447  
edcPropCtrl.TInspectorList 413  
edcPropCtrl.TInspectorList.Align 422  
edcPropCtrl.TInspectorList.Anchors 423  
edcPropCtrl.TInspectorList.BevelEdges 423  
edcPropCtrl.TInspectorList.BevelInner 423  
edcPropCtrl.TInspectorList.BevelKind 424  
edcPropCtrl.TInspectorList.BevelOuter 424  
edcPropCtrl.TInspectorList.BiDiMode 424  
edcPropCtrl.TInspectorList.BorderStyle 424  
edcPropCtrl.TInspectorList.ByCategories 425  
edcPropCtrl.TInspectorList.cCategories 425  
edcPropCtrl.TInspectorList.cDefValues 425  
edcPropCtrl.TInspectorList.cEditBackGround 425  
edcPropCtrl.TInspectorList.cEditValue 425  
edcPropCtrl.TInspectorList.cGutter 425  
edcPropCtrl.TInspectorList.cGutterBnd 425  
edcPropCtrl.TInspectorList.cHighlight 425  
edcPropCtrl.TInspectorList.cHighlightText 426  
edcPropCtrl.TInspectorList.Color 426  
edcPropCtrl.TInspectorList.Component 426  
edcPropCtrl.TInspectorList.Constraints 426  
edcPropCtrl.TInspectorList.cPropName 426  
edcPropCtrl.TInspectorList.cPropReadOnly 426  
edcPropCtrl.TInspectorList.cPropReference 427  
edcPropCtrl.TInspectorList.cPropValue 427  
edcPropCtrl.TInspectorList.cSubProperty 427  
edcPropCtrl.TInspectorList.Ctl3D 427  
edcPropCtrl.TInspectorList.DefPropNameDraw 427  
edcPropCtrl.TInspectorList.Designer 427  
edcPropCtrl.TInspectorList.DragCursor 428  
edcPropCtrl.TInspectorList.DragKind 428  
edcPropCtrl.TInspectorList.DragMode 428  
edcPropCtrl.TInspectorList.EditorVisible 428  
edcPropCtrl.TInspectorList.Enabled 428  
edcPropCtrl.TInspectorList.ExpandRefs 428  
edcPropCtrl.TInspectorList.FoldingIcon 429  
edcPropCtrl.TInspectorList.Font 429  
edcPropCtrl.TInspectorList.IncludeRefs 429  
edcPropCtrl.TInspectorList.ItemHeight 429  
edcPropCtrl.TInspectorList.ItemIndex 429  
edcPropCtrl.TInspectorList.Items 429  
edcPropCtrl.TInspectorList.MarkNonDefault 430  
edcPropCtrl.TInspectorList.OnAcceptCategory 430  
edcPropCtrl.TInspectorList.OnAcceptProperty 430  
edcPropCtrl.TInspectorList.OnCanResize 430  
edcPropCtrl.TInspectorList.OnChangeSelection 431  
edcPropCtrl.TInspectorList.OnClick 431  
edcPropCtrl.TInspectorList.OnConstrainedResize 431  
edcPropCtrl.TInspectorList.OnContextPopup 431  
edcPropCtrl.TInspectorList.OnDbClick 432  
edcPropCtrl.TInspectorList.OnDragDrop 432  
edcPropCtrl.TInspectorList.OnDragOver 432  
edcPropCtrl.TInspectorList.OnDrawPropCell 433



- edcPropCtrl.TInspectorList.OnEndDrag 433
- edcPropCtrl.TInspectorList.OnEnter 433
- edcPropCtrl.TInspectorList.OnExit 433
- edcPropCtrl.TInspectorList.OnGetCellParams 433
- edcPropCtrl.TInspectorList.OnGetPropReadOnly 433
- edcPropCtrl.TInspectorList.OnKeyDown 433
- edcPropCtrl.TInspectorList.OnKeyPress 434
- edcPropCtrl.TInspectorList.OnKeyUp 434
- edcPropCtrl.TInspectorList.OnMouseDown 434
- edcPropCtrl.TInspectorList.OnMouseMove 435
- edcPropCtrl.TInspectorList.OnMouseUp 435
- edcPropCtrl.TInspectorList.OnPropListUpdated 435
- edcPropCtrl.TInspectorList.OnResize 435
- edcPropCtrl.TInspectorList.OnSetPropValueA 435
- edcPropCtrl.TInspectorList.OnSetPropValueW 436
- edcPropCtrl.TInspectorList.OnStartDrag 436
- edcPropCtrl.TInspectorList.ParentBiDiMode 436
- edcPropCtrl.TInspectorList.ParentColor 436
- edcPropCtrl.TInspectorList.ParentCtl3D 436
- edcPropCtrl.TInspectorList.ParentFont 436
- edcPropCtrl.TInspectorList.ParentShowHint 437
- edcPropCtrl.TInspectorList.PopupListAlign 437
- edcPropCtrl.TInspectorList.PopupMenu 437
- edcPropCtrl.TInspectorList.ReadOnly 437
- edcPropCtrl.TInspectorList.ShowGrid 437
- edcPropCtrl.TInspectorList.ShowGutter 437
- edcPropCtrl.TInspectorList.ShowHint 437
- edcPropCtrl.TInspectorList.ShowReadOnly 437
- edcPropCtrl.TInspectorList.ShowSelFrame 438
- edcPropCtrl.TInspectorList.SplitPos 438
- edcPropCtrl.TInspectorList.TabOrder 438
- edcPropCtrl.TInspectorList.TabStop 438
- edcPropCtrl.TInspectorList.TopItem 438
- edcPropCtrl.TInspectorList.TypeKinds 438
- edcPropCtrl.TInspectorList.TypeSelector 438
- edcPropCtrl.TInspectorList.Visible 439
- edcPropCtrl.TOnInspSetPropValueEventA 447
- edcPropCtrl.TOnInspSetPropValueEventA type 447
- edcPropCtrl.TOnInspSetPropValueEventW 448
- edcPropCtrl.TOnInspSetPropValueEventW type 448
- edcPropCtrl.TPropertyNode 439
- edcPropCtrl.TPropertyNode.Create 441
- edcPropCtrl.TPropertyNode.Editor 442
- edcPropCtrl.TPropertyNode.Expandable 441
- edcPropCtrl.TPropertyNode.GetName 441
- edcPropCtrl.TPropertyNode.IsDefault 442
- edcPropCtrl.TPropertyNode.IsReference 442
- edcPropCtrl.TPropertyNode.IsSubProperty 442
- edcPropCtrl.TPropertyNode.Owner 441
- edcPropCtrl.TPropertyNode.PropInfo 442
- edcPropCtrl.TPropertyNode.ReflectModified 441
- edcPropCtrl.TPropertyNodes 442
- edcPropCtrl.TPropertyNodes.Clear 445
- edcPropCtrl.TPropertyNodes.Create 445
- edcPropCtrl.TPropertyNodes.Destroy 445
- edcPropCtrl.TPropertyNodes.ExpandItem 445
- edcPropCtrl.TPropertyNodes.GetProps 445
- edcPropCtrl.TTypeSelector 446
- edcPropCtrl.TTypeSelector enumeration 446
- edcPropCtrl.TzDesignerSelections 445
- edcPropEdit namespace
  - Classes 448
  - Interfaces 464
  - Types 465
- edcPropEdit.IPropertyStatusImage 464
- edcPropEdit.IPropertyStatusImage.DrawStatus 464
- edcPropEdit.IPropertyStatusImage.GetStatusWidth 464
- edcPropEdit.IPropertyStatusImage.StatusClick 465
- edcPropEdit.TCustomPropertyEdit 448
- edcPropEdit.TCustomPropertyEdit.AcceptListValue 453
- edcPropEdit.TCustomPropertyEdit.AcceptTab 455
- edcPropEdit.TCustomPropertyEdit.ButtonClick 453
- edcPropEdit.TCustomPropertyEdit.Change 453
- edcPropEdit.TCustomPropertyEdit.ChangePropertyValue 453
- edcPropEdit.TCustomPropertyEdit.Component 456
- edcPropEdit.TCustomPropertyEdit.Create 453
- edcPropEdit.TCustomPropertyEdit.DbClick 453
- edcPropEdit.TCustomPropertyEdit.Designer 456
- edcPropEdit.TCustomPropertyEdit.DoEdit 454
- edcPropEdit.TCustomPropertyEdit.DoExit 454
- edcPropEdit.TCustomPropertyEdit.DropDown 454
- edcPropEdit.TCustomPropertyEdit.GetEditor 454
- edcPropEdit.TCustomPropertyEdit.GetStr 454
- edcPropEdit.TCustomPropertyEdit.GetWStr 454

- edcPropEdit.TCustomPropertyEdit.MouseDown 454
- edcPropEdit.TCustomPropertyEdit.Notification 455
- edcPropEdit.TCustomPropertyEdit.OnSetPropValueA 457
- edcPropEdit.TCustomPropertyEdit.OnSetPropValueW 457
- edcPropEdit.TCustomPropertyEdit.PaintStatus 455
- edcPropEdit.TCustomPropertyEdit.PropertyEditor 456
- edcPropEdit.TCustomPropertyEdit.PropertyName 456
- edcPropEdit.TCustomPropertyEdit.ReadOnly 456
- edcPropEdit.TCustomPropertyEdit.SetValue 455
- edcPropEdit.TCustomPropertyEdit.TypeKinds 456
- edcPropEdit.TCustomPropertyEdit.UpdateEditState 455
- edcPropEdit.TOnSetPropValueEventA 465
- edcPropEdit.TOnSetPropValueEventA type 465
- edcPropEdit.TOnSetPropValueEventW 465
- edcPropEdit.TOnSetPropValueEventW type 465
- edcPropEdit.TPropertyEdit 457
- edcPropEdit.TPropertyEdit.Component 462
- edcPropEdit.TPropertyEdit.Designer 462
- edcPropEdit.TPropertyEdit.ListAlign 462
- edcPropEdit.TPropertyEdit.OnSetPropValueA 462
- edcPropEdit.TPropertyEdit.OnSetPropValueW 463
- edcPropEdit.TPropertyEdit.PropertyName 463
- edcPropEdit.TPropertyEdit.ReadOnly 463
- edcPropEdit.TPropertyEdit.TypeKinds 463
- edcPropEdit.TPropertyNameProperty 463
- edcToolList namespace
  - Classes 470
- edcToolList.TPaletteToolList 470
- edcToolList.TPaletteToolList.Align 477
- edcToolList.TPaletteToolList.AllowArrange 478
- edcToolList.TPaletteToolList.Anchors 478
- edcToolList.TPaletteToolList.AutoCollapse 478
- edcToolList.TPaletteToolList.BevelEdges 478
- edcToolList.TPaletteToolList.BevelInner 478
- edcToolList.TPaletteToolList.BevelKind 479
- edcToolList.TPaletteToolList.BevelOuter 479
- edcToolList.TPaletteToolList.BiDiMode 479
- edcToolList.TPaletteToolList.CategoryHeight 479
- edcToolList.TPaletteToolList.ClsChanged 476
- edcToolList.TPaletteToolList.ClsPalChanged 476
- edcToolList.TPaletteToolList.Color 479
- edcToolList.TPaletteToolList.ComponentAt 476
- edcToolList.TPaletteToolList.Constraints 480
- edcToolList.TPaletteToolList.Create 476
- edcToolList.TPaletteToolList.Ctl3D 480
- edcToolList.TPaletteToolList.CustomItems 480
- edcToolList.TPaletteToolList.Destroy 477
- edcToolList.TPaletteToolList.DragCursor 481
- edcToolList.TPaletteToolList.DragImageType 481
- edcToolList.TPaletteToolList.DragKind 481
- edcToolList.TPaletteToolList.DragMode 481
- edcToolList.TPaletteToolList.DrawItemImage 477
- edcToolList.TPaletteToolList.Enabled 481
- edcToolList.TPaletteToolList.Filtered 481
- edcToolList.TPaletteToolList.FilterString 482
- edcToolList.TPaletteToolList.FoldingIcon 482
- edcToolList.TPaletteToolList.Font 482
- edcToolList.TPaletteToolList.HintProps 482
- edcToolList.TPaletteToolList.ItemHeight 482
- edcToolList.TPaletteToolList.ItemIndexChanged 477
- edcToolList.TPaletteToolList.Items 482
- edcToolList.TPaletteToolList.ItemsArranged 477
- edcToolList.TPaletteToolList.OnCanResize 482
- edcToolList.TPaletteToolList.OnClick 483
- edcToolList.TPaletteToolList.OnConstrainedResize 483
- edcToolList.TPaletteToolList.OnContextPopup 483
- edcToolList.TPaletteToolList.OnDbClick 484
- edcToolList.TPaletteToolList.OnDragDrop 484
- edcToolList.TPaletteToolList.OnDragOver 484
- edcToolList.TPaletteToolList.OnEndDrag 484
- edcToolList.TPaletteToolList.OnEnter 484
- edcToolList.TPaletteToolList.OnExit 485
- edcToolList.TPaletteToolList.OnKeyDown 485
- edcToolList.TPaletteToolList.OnKeyPress 485
- edcToolList.TPaletteToolList.OnKeyUp 485
- edcToolList.TPaletteToolList.OnMouseDown 486
- edcToolList.TPaletteToolList.OnMouseMove 486
- edcToolList.TPaletteToolList.OnMouseUp 486
- edcToolList.TPaletteToolList.OnPalChange 489
- edcToolList.TPaletteToolList.OnResize 486
- edcToolList.TPaletteToolList.OnStartDrag 487
- edcToolList.TPaletteToolList.ParentBiDiMode 487
- edcToolList.TPaletteToolList.ParentColor 487
- edcToolList.TPaletteToolList.ParentCtl3D 487

- edcToolList.TPaletteToolList.ParentFont 487
- edcToolList.TPaletteToolList.ParentShowHint 488
- edcToolList.TPaletteToolList.PopupMenu 488
- edcToolList.TPaletteToolList.RowSpace 488
- edcToolList.TPaletteToolList.ShowCaptions 488
- edcToolList.TPaletteToolList.ShowCategory 477
- edcToolList.TPaletteToolList.ShowHint 488
- edcToolList.TPaletteToolList.StyleCategory 488
- edcToolList.TPaletteToolList.StyleCategoryMouseOver 488
- edcToolList.TPaletteToolList.StyleCategorySelected 488
- edcToolList.TPaletteToolList.StyleItem 488
- edcToolList.TPaletteToolList.StyleItemMouseOver 488
- edcToolList.TPaletteToolList.StyleItemSelected 488
- edcToolList.TPaletteToolList.TabOrder 489
- edcToolList.TPaletteToolList.TabStop 489
- edcToolList.TPaletteToolList.TransparentImages 489
- edcToolList.TPaletteToolList.VerticalGroups 489
- edcToolList.TPaletteToolList.Visible 489
- eddAlignDlg namespace
  - Classes 465
- eddAlignDlg.TAlignmentDlg 465
- eddAlignPal namespace
  - Classes 466
- eddAlignPal.TAlignPalette 466
- eddCrOrdDI namespace
  - Classes 467
- eddCrOrdDI.TCreateOrderDlg 468
- eddCustomPal namespace
  - Classes 468
- eddCustomPal.TCustomizePaletteDlg 468
- eddDsnOpt namespace
  - Classes 490
- eddDsnOpt.TDsnOptionsDlg 490
- eddDsnOpt.TDsnOptionsDlg.Designer 491
- eddObjInspFrm namespace
  - Classes 494
  - Types 498
- eddObjInspFrm.TObjectInspectorFrame 495
- eddObjInspFrm.TObjectInspectorFrame.ComponentCombo 497
- eddObjInspFrm.TObjectInspectorFrame.Create 496
- eddObjInspFrm.TObjectInspectorFrame.Customize 497
- eddObjInspFrm.TObjectInspectorFrame.EventsList 497
- eddObjInspFrm.TObjectInspectorFrame.IntegralHeight 497
- eddObjInspFrm.TObjectInspectorFrame.OnHideClick 498
- eddObjInspFrm.TObjectInspectorFrame.OnStayOnTopClick 498
- eddObjInspFrm.TObjectInspectorFrame.PageControl 497
- eddObjInspFrm.TObjectInspectorFrame.Pages 497
- eddObjInspFrm.TObjectInspectorFrame.PropertyList 497
- eddObjInspFrm.TObjectInspectorFrame.ReadOnly 497
- eddObjInspFrm.TObjectInspectorFrame.ShowInstanceList 497
- eddObjInspFrm.TObjectInspectorFrame.ShowStatusBar 497
- eddObjInspFrm.TObjInspTabs 498
- eddObjInspFrm.TObjInspTabs type 498
- eddObjInspProp namespace
  - Classes 491
- eddObjInspProp.TObjInspPropDlg 491
- eddObjInspProp.TObjInspPropDlg.ObjectInspector 493
- eddObjTreeFrame namespace
  - Classes 500
- eddObjTreeFrame.TObjectTreeFrame 500
- eddObjTreeFrame.TObjectTreeFrame.Create 500
- eddObjTreeFrame.TObjectTreeFrame.ObjectTree 501
- eddPackageCtrl namespace
  - Classes 493
- eddPackageCtrl.TPackageCtrlDlg 493
- eddPageName namespace
  - Classes 494
- eddPageName.TPageNameDlg 494
- eddScaleDI namespace
  - Classes 498
- eddScaleDI.TScaleDlg 498
- eddSelFrame namespace
  - Classes 499
- eddSelFrame.TSelFrameDlg 499
- eddSizeDlg namespace
  - Classes 501
- eddSizeDlg.TSizeAdjDlg 501
- eddTabOrdDI namespace
  - Classes 502
- eddTabOrdDI.TTabOrderDlg 502
- edlOUtils namespace
  - Functions 503
- edlOUtils.zCopyCmpResource 504

- edlOUtils.zCopyCmpResource function 504
- edlOUtils.zReadCmpFromFile 504
- edlOUtils.zReadCmpFromFile function 504
- edlOUtils.zReadCmpFromStream 504
- edlOUtils.zReadCmpFromStream function 504
- edlOUtils.zWriteCmpToFile 505
- edlOUtils.zWriteCmpToFile function 505
- edlOUtils.zWriteCmpToStream 505
- edlOUtils.zWriteCmpToStream function 505
- edManager namespace
  - Classes 505
  - Functions 513
  - Interfaces 511
  - Structs, Records, Enums 513
  - Variables 514
- edManager.DsnManager 514
- edManager.DsnManager variable 514
- edManager.GetClassDragImage 513
- edManager.GetClassDragImage function 513
- edManager.IClassSelector 511
- edManager.IClassSelector.ClsChanged 511
- edManager.IClassSelector.ClsPalChanged 512
- edManager.IDesignIDEEEvents 512
- edManager.IDesignIDEEEvents.ActiveDsnChanged 512
- edManager.IDesignIDEEEvents.BeforeRegisterComponent 512
- edManager.IDesignIDEEEvents.GetGlobalComponents 512
- edManager.IDesignIDEEEvents.GetWorkspaceOrigin 513
- edManager.IDesignIDEEEvents.KeyDown 513
- edManager.IDesignIDEEEvents.KeyPress 513
- edManager.TComponentClassDragImage 513
- edManager.TComponentClassDragImage enumeration 513
- edManager.TDesignerManager 505
- edManager.TDesignerManager.ActiveDesigner 510
- edManager.TDesignerManager.AddClient 507
- edManager.TDesignerManager.BeforeRegisterComponent 507
- edManager.TDesignerManager.ComponentClass 510
- edManager.TDesignerManager.Create 507
- edManager.TDesignerManager.CreateCurrent 507
- edManager.TDesignerManager.DesignerClosed 507
- edManager.TDesignerManager.DesignerOpened 508
- edManager.TDesignerManager.Destroy 508
- edManager.TDesignerManager.GetGlobalComponents 508
- edManager.TDesignerManager.GetWorkspaceOrigin 508
- edManager.TDesignerManager.ItemDeleted 508
- edManager.TDesignerManager.ItemInserted 508
- edManager.TDesignerManager.ItemsModified 509
- edManager.TDesignerManager.KeyDown 509
- edManager.TDesignerManager.KeyPress 509
- edManager.TDesignerManager.MultiCreate 511
- edManager.TDesignerManager.PaletteChanged 509
- edManager.TDesignerManager.RemoveClient 509
- edManager.TDesignerManager.ResetCmpClass 509
- edManager.TDesignerManager.SelectionChanged 509
- edManager.TDesignerManager.SetActiveDesigner 510
- edsMenuDsn namespace
  - Classes 514
- edsMenuDsn.TMenuDsnWnd 514
- edsMenuDsn.TzMenuEditor 515
- edsMenuDsn.TzMenuItemsPropertyEditor 515
- eduDMContainer namespace
  - Classes 515
- eduDMContainer.TDsnDM 516
- eduServObj namespace
  - Classes 516
  - Functions 523
  - Structs, Records, Enums 523
- eduServObj.DrawPatternRect 523
- eduServObj.DrawPatternRect function 523
- eduServObj.IsServiceControl 523
- eduServObj.IsServiceControl function 523
- eduServObj.TAlignRuler 516
- eduServObj.TComponentCaption 517
- eduServObj.TComponentIcon 517
- eduServObj.TDraggedControl 517
- eduServObj.TMarkerShape 524
- eduServObj.TMarkerShape enumeration 524
- eduServObj.TSmallRect 517
- eduServObj.TTabOrderIcons 518
- eduServObj.TTabOrderIcons.Color 519
- eduServObj.TTabOrderIcons.Font 519
- eduServObj.TTabOrderIcons.Height 519
- eduServObj.TTabOrderIcons.Hide 519
- eduServObj.TTabOrderIcons.HorzAlign 519
- eduServObj.TTabOrderIcons.SetTabOrder 519

- eduServObj.TTabOrderIcons.Show 519
- eduServObj.TTabOrderIcons.VertAlign 519
- eduServObj.TTabOrderIcons.Visible 519
- eduServObj.TTabOrderIcons.Width 520
- eduServObj.TVerticalAlignment 524
- eduServObj.TVerticalAlignment enumeration 524
- eduServObj.TzBoundCtrl 520
- eduServObj.TzBoundCtrl.Bitmap 522
- eduServObj.TzBoundCtrl.Color 522
- eduServObj.TzBoundCtrl.Control 522
- eduServObj.TzBoundCtrl.DrawMultSel 522
- eduServObj.TzBoundCtrl.GrabAtPos 521
- eduServObj.TzBoundCtrl.GrabSize 522
- eduServObj.TzBoundCtrl.Hide 521
- eduServObj.TzBoundCtrl.Invalidate 521
- eduServObj.TzBoundCtrl.Local 522
- eduServObj.TzBoundCtrl.Locked 522
- eduServObj.TzBoundCtrl.MarkerShape 522
- eduServObj.TzBoundCtrl.Recreate 521
- eduServObj.TzBoundCtrl.Update 521
- eduServObj.TzBoundCtrl.Visible 523
- edUtils namespace
  - Functions 524
  - Structs, Records, Enums 527
- edUtils.DsnAlignSelected 525
- edUtils.DsnAlignSelected function 525
- edUtils.DsnLoadPackage 525
- edUtils.DsnLoadPackage function 525
- edUtils.DsnReadCmpFromStream 525
- edUtils.DsnReadCmpFromStream function 525
- edUtils.DsnReadFromFile 525
- edUtils.DsnReadFromFile function 525
- edUtils.DsnWriteCmpToStream 526
- edUtils.DsnWriteCmpToStream function 526
- edUtils.DsnWriteToFile 526
- edUtils.DsnWriteToFile function 526
- edUtils.GetDesigner 526
- edUtils.GetDesigner function 526
- edUtils.InvalidateControl 526
- edUtils.InvalidateControl function 526
- edUtils.IsControlParent 526
- edUtils.IsControlParent function 526

- edUtils.NormalizeRect 527
- edUtils.NormalizeRect function 527
- edUtils.PerformDsnAction 527
- edUtils.PerformDsnAction function 527
- edUtils.ShowDesignerOptionsDlg 527
- edUtils.ShowDesignerOptionsDlg function 527
- edUtils.ShowDsnAbout 527
- edUtils.ShowDsnAbout function 527
- edUtils.TDesignerAction 528
- edUtils.TDesignerAction enumeration 528
- event handlers; methods 10
- Examples 9

## F

- Features 2
- Frames class 242

## I

- IClassSelector interface 511
  - ClsChanged 511
  - ClsPalChanged 512
- IDesignIDEEEvents interface 512
  - ActiveDsnChanged 512
  - BeforeRegisterComponent 512
  - GetGlobalComponents 512
  - GetWorkspaceOrigin 513
  - KeyDown 513
  - KeyPress 513
- Insallation 3
- Integration with scripters 6
- IPropertyStatusImage interface 464
  - DrawStatus 464
  - GetStatusWidth 464
  - StatusClick 465

## L

- License 8

## O

- Overview 1

## R

Registration Method Example 10

## T

TAlignmentDlg class 465

TAlignPalette class 466

TAlignRuler class 516

TBaseDesigner class 144

Active 151

CanProcessNCMessages 146

Client2Screen 147

ClientOrg 147

Create 147

DesignState 147

Destroy 147

DoObjectHint 147

DragDrop 147

DragOver 148

HintObject 152

IsRTL 148

KeyDown 148

KeyPress 149

KeyUp 149

Loaded 149

MouseDown 150

MouseMove 150

MouseUp 150

OnActiveChanged 152

OnDragDrop 152

OnDragOver 152

OnHandleControlMessage 153

OnKeyDown 153

OnKeyPress 153

OnKeyUp 154

OnMouseDown 154

OnMouseMove 154

OnMouseUp 154

ProcessMessage 151

ResetHint 151

Screen2Client 151

SetActive 151

ShowHint 151

ShowHints 152

TBtnEdit class 70

AdjustClientRect 73

Alignment 76

ButtonClick 73

ButtonVisible 76

ButtonWidth 77

Canvas 77

Create 73

CreateParams 73

CreateWnd 73

Destroy 74

EndTracking 74

KeyDown 74

KeyPress 74

MouseMove 74

MouseUp 75

MultiLine 77

OnButtonClick 78

Paint 75

PaintBtnGlyph 75

PaintStatus 75

PaintWindow 75

PtInButton 75

StartTracking 76

StatusWidth 77

StopTracking 76

TrackButton 76

WantReturns 77

WantTabs 78

WordWrap 78

TBtnMargins class 18

Bottom 19

BtnHorz 19

BtnVert 19

Left 19

Right 20

Top 20

TBtnMargins.Margins example 10

TBtnPanel class 28

Align 32

Anchors 33	ParentBiDiMode 42
AutoSize 33	ParentColor 42
BevelInner 33	ParentCtl3D 43
BevelOuter 33	ParentFont 43
BevelWidth 34	ParentShowHint 43
BiDiMode 34	PopupMenu 43
BorderStyle 34	RowCount 43
BorderWidth 34	ShowHint 43
ButtonCount 35	TabOrder 43
ButtonHeight 35	TabStop 43
ButtonWidth 35	Transparent 44
Color 35	Visible 44
Constraints 36	TCategoryNode class 396
Ctl3D 36	Clear 398
DownButton 36	Create 398
DragCursor 36	Expandable 398
DragKind 36	GetName 399
DragMode 37	HasValue 399
Enabled 37	TComponentCaption class 517
Flat 37	TComponentClassInfo class 242
Font 37	ComponentClass 244
HintProps 37	Create 243
Margins 37	Destroy 243
OnButtonClick 38	DisplayName 244
OnCanResize 38	Hidden 244
OnClick 38	Icon 244
OnConstrainedResize 38	InitPage 244
OnContextPopup 39	IsIconValid 244
OnDbClick 39	Module 244
OnDragDrop 39	Page 245
OnDragOver 39	TComponentCombo class 341
OnDrawButton 40	Align 346
OnEndDrag 40	Anchors 346
OnEnter 40	AutoCloseUp 347
OnExit 40	AutoDropDown 347
OnGetButtonHint 41	AutoHint 347
OnMouseDown 41	BiDiMode 347
OnMouseMove 41	Change 344
OnMouseUp 41	ClassNameColor 347
OnResize 42	ClassNameDelim 348
OnStartDrag 42	Color 348
Orientation 42	Constraints 348

Create 344	OnStartDrag 355
Ctl3D 348	ParentBiDiMode 356
Designer 348	ParentColor 356
Destroy 345	ParentCtl3D 356
DoAddObject 345	ParentFont 356
DragCursor 348	ParentShowHint 356
DragKind 349	PopupMenu 356
DragMode 349	SetSelection 345
DropDownCount 349	ShowClassName 356
DropDownWidth 349	ShowComponents 357
Enabled 349	ShowHint 357
FillObjList 345	Sorted 357
Font 350	TabOrder 357
ImeMode 350	TabStop 357
ImeName 350	Text 357
IncludeContainer 350	UpdateObjectList 345
ItemHeight 350	Visible 358
MaxLength 351	TComponentIcon class 517
NameColor 351	TControlGroups class 156
Notification 345	Clear 158
OnCanAddObject 358	Count 158
OnClick 351	Create 158
OnCloseUp 351	Destroy 158
OnContextPopup 351	GroupControls 158
OnDbClick 352	GroupForControl 158
OnDragDrop 352	Groups 158
OnDragOver 352	GroupSelected 158
OnDrawItem 353	UnGroup 158
OnDropDown 353	UnGroupSelected 158
OnEndDock 353	TCreateOrderDlg class 468
OnEndDrag 353	TCustomBtnPanel class 20
OnEnter 353	AutoSize 25
OnExit 354	ButtonAtPos 22
OnGetClassName 358	ButtonClick 22
OnGetComponents 358	ButtonCount 25
OnKeyDown 354	ButtonHeight 25
OnKeyPress 354	ButtonRect 22
OnKeyUp 354	ButtonWidth 25
OnMeasureItem 355	CanAutoSize 23
OnSelChanged 358	Caption 25
OnSelect 355	Create 23
OnStartDock 355	Destroy 23



DownButton 26	EndTracking 83
DrawButton 23	KeyPress 83
Flat 26	ListAlign 84
GetButtonHint 24	MouseDown 83
HintProps 26	MouseMove 83
InvalidateButtons 24	OnAcceptListValue 85
Loaded 24	OnCloseUp 85
Margins 26	OnDropDown 85
MouseDown 24	OnMeasureWidth 85
OnButtonClick 27	PaintBtnGlyph 84
OnDrawButton 27	PickList 84
OnGetButtonHint 28	StartTracking 84
Orientation 26	TCustomInspectorList class 399
Paint 24	AcceptProperty 404
RowCount 27	ByCategories 407
Transparent 27	Categories 408
TCustomDesignerObjTree class 267	cCategories 408
AddSprigAddItems 269	cDefValues 408
AddType 269	cEditBackGround 408
AddTypeCount 269	cEditValue 408
AddTypes 271	Component 408
CanDelete 269	CopyName 404
CanMove 269	CopyValue 405
Create 269	cPropName 409
DeleteSelected 270	cPropReadOnly 409
Designer 271	cPropReference 409
Destroy 270	cPropValue 409
Loaded 270	Create 405
Move 270	CreateEditor 405
Notification 270	CreateItems 405
OnCreateSprigNode 271	cSubProperty 409
RootSprig 271	CutValue 405
TCustomEditEx class 78	DefPropNameDraw 409
AcceptListValue 82	Designer 409
ActiveList 84	DoPrepareCanvas 405
ButtonClick 82	DrawPropCell 406
CloseUp 82	EditedObject 410
Create 82	ExpandRefs 410
Destroy 82	FocusEditor 406
DoDropDownKeys 83	GetDesigner 406
DropDown 83	HiddenCount 410
EditStyle 84	HintProps 410

IncludeRefs 410	DoExit 454
IsPropReadOnly 406	DropDown 454
KeyDown 406	GetEditor 454
Loaded 406	GetStr 454
MarkNonDefault 410	GetWStr 454
MouseDown 406	MouseDown 454
OnAcceptCategory 412	Notification 455
OnAcceptProperty 412	OnSetPropValueA 457
OnChangeSelection 412	OnSetPropValueW 457
OnGetPropReadOnly 413	PaintStatus 455
OnPropListUpdated 413	PropertyEditor 456
OnPropValueChanged 413	PropertyName 456
OnSetPropValueA 413	ReadOnly 456
OnSetPropValueW 413	SetValue 455
PasteValue 406	TypeKinds 456
PopupListAlign 411	UpdateEditState 455
PropValueChanged 407	TCustomPropList class 46
ReadOnly 411	cGutter 50
SaveValue 407	cGutterBnd 51
SearchPropKey 411	cHighlight 51
SearchPropMode 411	cHighlightText 51
Selected 411	Create 49
SelectedCount 411	CreateItems 49
SetItemIndex 407	Current 49
ShowReadOnly 411	Destroy 49
TypeKinds 411	DoPrepareCanvas 50
TypeSelector 412	DrawCell 50
UpdateEditor 407	DrawPropCell 50
UpdateList 407	FoldingIcon 51
TCustomizePaletteDlg class 468	GutterWidth 50
TCustomModuleInfo class 245	IsHeaderItem 50
TCustomPropertyEdit class 448	Items 51
AcceptListValue 453	LeftMargin 51
AcceptTab 455	LevelWidth 51
ButtonClick 453	MouseDown 50
Change 453	OnDrawPropCell 51
ChangePropertyValue 453	OnGetCellParams 52
Component 456	ShowGutter 51
Create 453	TCustomToolList class 116
DbClick 453	AllowArrange 120
Designer 456	AutoCollapse 120
DoEdit 454	CategoryHeight 120

CollapseAll 118	AddMethod 264
Create 119	Count 265
Destroy 119	Create 264
DrawItemImage 119	Destroy 264
ExpandAll 119	FInfos 261
Filtered 120	GetMethodsNames 264
FilterString 121	Items 265
FoldingIcon 121	RemoveObject 265
GetCategoryItem 119	ValidateMethod 265
HintProps 121	TDesignerAction class 308
Images 121	Caption 309
InsertAtItem 121	Enabled 309
ItemAtPos 119	HelpContext 309
ItemHeight 121	HelpKeyword 310
ItemIndex 121	HelpType 310
ItemIndexChanged 119	Hint 310
ItemRect 119	ImageIndex 310
Items 121	OnExecute 310
ItemsArranged 120	OnHint 310
ItemsChanged 120	OnUpdate 311
ItemsHeight 120	SecondaryShortCuts 311
MakeTopItem 120	ShortCut 311
MakeVisible 120	Update 309
MouseOverItem 122	Visible 311
OnItemArranged 123	TDesignerEvents class 389
OnItemChanged 123	OnActiveDsnChanged 391
PaintItem 120	OnClassChanged 391
RightClickSelect 122	OnDesignerClosed 391
RowSpace 122	OnDesignerInitialized 391
Selected 122	OnDsnKeyDown 392
SelectFirstVisible 120	OnGetGlobalComponents 392
StyleCategory 122	OnGetWorkspaceOrigin 392
StyleCategoryMouseOver 122	OnItemDeleted 392
StyleCategorySelected 122	OnItemInserted 393
StyleItem 122	OnItemsModified 393
StyleItemMouseOver 122	OnKeyPress 393
StyleItemSelected 122	OnPaletteChanged 393
VerticalGroups 122	OnRegisterComponent 394
ViewOrigin 123	OnSelectionChanged 394
TDefaultMethodRegister class 259	TDesignerManager class 505
Add 261, 262, 263, 264	ActiveDesigner 510
Add method 261	AddClient 507

BeforeRegisterComponent 507	HideSelection 281
ComponentClass 510	HotTrack 281
Create 507	Images 282
CreateCurrent 507	Indent 282
DesignerClosed 507	Items 282
DesignerOpened 508	MultiSelect 282
Destroy 508	MultiSelectStyle 282
GetGlobalComponents 508	OnAddition 283
GetWorkspaceOrigin 508	OnAdvancedCustomDraw 283
ItemDeleted 508	OnAdvancedCustomDrawItem 283
ItemInserted 508	OnChange 283
ItemsModified 509	OnChanging 284
KeyDown 509	OnClick 284
KeyPress 509	OnCollapsed 284
MultiCreate 511	OnCollapsing 284
PaletteChanged 509	OnCompare 284
RemoveClient 509	OnContextPopup 285
ResetCmpClass 509	OnCreateNodeClass 285
SelectionChanged 509	OnCreateSprigNode 285
SetActiveDesigner 510	OnCustomDraw 285
TDesignerObjTree class 271	OnCustomDrawItem 286
Align 277	OnDbClick 286
Anchors 277	OnDeletion 286
AutoExpand 278	OnDragDrop 286
BevelEdges 278	OnDragOver 286
BevelInner 278	OnEdited 287
BevelKind 278	OnEditing 287
BevelOuter 279	OnEndDock 287
BevelWidth 279	OnEndDrag 287
BiDiMode 279	OnEnter 288
BorderStyle 279	OnExit 288
BorderWidth 279	OnExpanded 288
ChangeDelay 280	OnExpanding 288
Color 280	OnGetImageIndex 288
Constraints 280	OnGetSelectedIndex 288
Create 277	OnKeyDown 289
Ctl3D 280	OnKeyPress 289
DragCursor 280	OnKeyUp 289
DragKind 281	OnMouseDown 289
DragMode 281	OnMouseMove 290
Enabled 281	OnMouseUp 290
Font 281	OnStartDock 290

OnStartDrag 291	TdsnBringToFront class 315
ParentBiDiMode 291	Execute 316
ParentColor 291	TdsnCopy class 316
ParentCtl3D 291	TdsnCreationOrderDlg class 317
ParentFont 291	Execute 318
ParentShowHint 291	TdsnCut class 318
PopupMenu 292	TdsnDelete class 318
ReadOnly 292	TdsnDesignMode class 319
RightClickSelect 292	Execute 320
RowSelect 292	Update 320
ShowButtons 292	TDsnDM class 516
ShowHint 293	TdsnFlipChildren class 320
ShowLines 293	Execute 322
ShowRoot 293	TdsnFlipChildrenAll class 322
SortType 293	Execute 323
StateImages 293	TdsnGroupControls class 323
TabOrder 293	Execute 325
TabStop 293	Update 325
ToolTips 294	TDsnInplaceEditor class 295
Visible 294	Adapter 300
TDesignSurface class 237	Alignment 300
Activate 238	Close 299
AdjustScroll 238	Color 300
DoSizing 238	Create 299
DsnShowFrame 239	Destroy 300
ExecuteAction 238	IsUnicode 301
FlatScrollBars 239	LoadText 300
Form 240	MultiLine 301
FormOrigin 240	OnChange 301
FrameSize 240	OnExit 301
HideFormBorders 240	SaveText 300
RulerClientArea 240	TextW 301
ScrollPos 240	WordWrap 302
ShowFrame 240	TdsnLockControls class 325
ShowRuler 241	Execute 326
UpdateAction 239	Update 326
UseUnits 241	TDsnOptionsDlg class 490
TDraggedControl class 517	Designer 491
TdsnAlignmentDlg class 311	TdsnPaste class 327
Execute 313	TdsnRedo class 327
TdsnAlignToGrid class 313	TdsnScale class 327
Execute 314	Execute 328

TDsnSelAction class 329	MouseToItem 57
Update 330	MouseUp 57
TdsnSelectAll class 330	OnClick 60
TdsnSendToBack class 330	Paint 57
Execute 332	SetItemIndex 57
TdsnShowTabOrder class 332	ShowGrid 60
Execute 333	ShowSelfFrame 60
Update 333	SplitPos 60
TdsnSizeDlg class 334	TabOrder 60
Execute 335	TabStop 60
TdsnTabOrderDlg class 335	TopItem 60
Execute 336	UpdateAction 57
TdsnTargetAction class 337	UpdateEditor 58
TdsnTextEditMode class 337	TechHintHelper class 111
Execute 338	CancelHint 112
Update 338	CanMoveLeft 113
TdsnUndo class 338	Color 113
TdsnUngroupControls class 339	ControlWndProc 112
Execute 340	Create 112
TDualList class 52	Destroy 113
BorderStyle 58	Enabled 113
Canvas 58	Font 113
Create 54	HidePause 114
CreateEditor 54	Pause 114
CreateHandle 54	ResetHint 113
Destroy 54	ShortPause 114
DoMouseWheel 54	ShowHint 113
DrawCell 55	TEditEx class 85
DrawStr 55	Alignment 92
DrawStrW 55	Anchors 92
Editor 59	AutoSelect 92
EditorVisible 59	AutoSize 92
ExecuteAction 55	BevelEdges 93
FocusEditor 56	BevelInner 93
IsHeaderItem 56	BevelKind 93
ItemCount 59	BevelOuter 93
ItemHeight 59	BevelWidth 93
ItemIndex 59	BiDiMode 94
ItemRect 56	BorderStyle 94
KeyDown 56	ButtonWidth 94
MouseDown 56	CharCase 94
MouseMove 56	Color 95

Constraints 95	ReadOnly 103
Ctl3D 95	SetTextW 103
DragCursor 95	ShowHint 104
DragKind 95	StatusWidth 104
DragMode 95	TabOrder 104
EditMask 96	TabStop 104
EditStyle 96	Text 104
Enabled 96	TextW 104
Font 96	Visible 104
ImeMode 96	TFrameInfo class 245
ImeName 97	FrameClass 246
IsUnicode 97	FrameResource 246
ListAlign 97	TInplaceComponentEditor class 302
MaxLength 97	BoundRect 305
OnAcceptListValue 98	Control 305
OnButtonClick 98	Create 303
OnChange 98	GetBoundRect 303
OnClick 98	GetText 304
OnCloseUp 98	GetTextW 304
OnDbClick 98	HandlePos 304
OnDragDrop 98	IsAutoUpdate 304
OnDragOver 99	IsUnicode 304
OnDropDown 99	SetEditor 304
OnEndDrag 99	SetHitPoint 305
OnEnter 99	SetText 305
OnExit 100	SetTextW 305
OnKeyDown 100	Text 305
OnKeyPress 100	TextW 306
OnKeyUp 100	TInspectorList class 413
OnMeasureWidth 101	Align 422
OnMouseDown 101	Anchors 423
OnMouseMove 101	BevelEdges 423
OnMouseUp 101	BevelInner 423
OnStartDrag 102	BevelKind 424
ParentBiDiMode 102	BevelOuter 424
ParentColor 102	BiDiMode 424
ParentCtl3D 102	BorderStyle 424
ParentFont 102	ByCategories 425
ParentShowHint 102	cCategories 425
PasswordChar 103	cDefValues 425
PickList 103	cEditBackGround 425
PopupMenu 103	cEditValue 425

cGutter 425	OnGetCellParams 433
cGutterBnd 425	OnGetPropReadOnly 433
cHighlight 425	OnKeyDown 433
cHighlightText 426	OnKeyPress 434
Color 426	OnKeyUp 434
Component 426	OnMouseDown 434
Constraints 426	OnMouseMove 435
cPropName 426	OnMouseUp 435
cPropReadOnly 426	OnPropListUpdated 435
cPropReference 427	OnResize 435
cPropValue 427	OnSetPropValueA 435
cSubProperty 427	OnSetPropValueW 436
Ctl3D 427	OnStartDrag 436
DefPropNameDraw 427	ParentBiDiMode 436
Designer 427	ParentColor 436
DragCursor 428	ParentCtl3D 436
DragKind 428	ParentFont 436
DragMode 428	ParentShowHint 437
EditorVisible 428	PopupListAlign 437
Enabled 428	PopupMenu 437
ExpandRefs 428	ReadOnly 437
FoldingIcon 429	ShowGrid 437
Font 429	ShowGutter 437
IncludeRefs 429	ShowHint 437
ItemHeight 429	ShowReadOnly 437
ItemIndex 429	ShowSelfFrame 438
Items 429	SplitPos 438
MarkNonDefault 430	TabOrder 438
OnAcceptCategory 430	TabStop 438
OnAcceptProperty 430	TopItem 438
OnCanResize 430	TypeKinds 438
OnChangeSelection 431	TypeSelector 438
OnClick 431	Visible 439
OnConstrainedResize 431	TMenuDsnWnd class 514
OnContextPopup 431	TObjectInspectorFrame class 495
OnDbClick 432	ComponentCombo 497
OnDragDrop 432	Create 496
OnDragOver 432	Customize 497
OnDrawPropCell 433	EventsList 497
OnEndDrag 433	IntegralHeight 497
OnEnter 433	OnHideClick 498
OnExit 433	OnStayOnTopClick 498



- PageControl 497
- Pages 497
- PropertyList 497
- ReadOnly 497
- ShowInstanceList 497
- ShowStatusBar 497
- TObjectTreeFrame class 500
  - Create 500
  - ObjectTree 501
- TObjInspPropDlg class 491
  - ObjectInspector 493
- TPackageCtrlDlg class 493
- TPackageInfo class 246
  - Active 247
  - Create 247
  - Description 247
  - Destroy 247
  - FileName 247
  - Handle 247
  - Requires 247
  - Units 248
- TPackageMng class 248
  - AddComponent 250
  - AddPackage 250
  - AutoSave 255
  - BeginUpdate 250
  - ComponentCount 255
  - Components 255
  - Create 250
  - CreateFrame 251
  - CustomizePackages 251
  - CustomizePalette 251
  - DeleteComponent 251
  - DeleteComponent method 251
  - Destroy 251
  - EndUpdate 252
  - FindClass 252
  - FindClassName 252
  - FindClassNameldx 252
  - FindPackage 252
  - FrameInfos 255
  - GetCustomModule 252
  - IsNolcon 253
  - LoadPaletteFromIni 253
  - OnRegisterComponent 256
  - OnRegisterComponentInfo 256
  - OnUnRegisterComponentInfo 256
  - Packages 255
  - Pages 256
  - ReadRegInfo 253
  - RegisterFrame 253
  - RegSubkey 256
  - RemoveEmptyPages 253
  - RemovePackage 254
  - RenamePage 254
  - ResetPalette 254
  - SavePaletteToIni 254
  - SaveRegInfo 254
  - SetComponentOrder 255
- TPageNameDlg class 494
- TPalettePanel class 360
  - Align 366
  - Anchors 366
  - AutoSize 367
  - BevelInner 367
  - BevelOuter 367
  - BevelWidth 367
  - BiDiMode 368
  - BorderStyle 368
  - BorderWidth 368
  - ButtonClick 364
  - ButtonHeight 368
  - ButtonWidth 369
  - Color 369
  - Constraints 369
  - Create 365
  - Ctl3D 369
  - Destroy 365
  - DownButton 370
  - DragCursor 370
  - DragImageType 370
  - DragKind 370
  - DragMode 370
  - DrawButton 365

Enabled 370	DragCursor 381
Flat 371	DragKind 381
Font 371	DragMode 381
GetButtonHint 365	Enabled 381
HintProps 371	Flat 382
Margins 371	Font 382
OnButtonClick 371	HintProps 382
OnCanResize 372	HotTrack 382
OnClick 372	Images 382
OnConstrainedResize 372	MultiLine 382
OnContextPopup 372	OnChange 383
OnDbClick 373	OnChanging 383
OnDragDrop 373	OnContextPopup 383
OnDragOver 373	OnDragDrop 383
OnEndDrag 374	OnDragOver 383
OnEnter 374	OnDrawTab 383
OnExit 374	OnEndDock 384
OnMouseDown 374	OnEndDrag 384
OnMouseMove 374	OnEnter 384
OnMouseUp 375	OnExit 384
OnResize 375	OnGetImageIndex 384
OnStartDrag 375	OnMouseDown 384
Orientation 375	OnMouseMove 385
Page 376	OnMouseUp 385
ParentBiDiMode 376	OnResize 385
ParentColor 376	OnStartDrag 385
ParentCtl3D 376	OwnerDraw 385
ParentFont 376	PalettePanel 386
ParentShowHint 376	ParentBiDiMode 386
PopupMenu 376	ParentFont 386
RowCount 376	ParentShowHint 386
ShowHint 377	PopupMenu 386
TabOrder 377	RaggedRight 386
TabStop 377	ResetOnChange 386
Transparent 377	ScrollOpposite 387
UpdateList 366	ShowHint 387
Visible 377	Style 387
TPaletteTab class 377	TabHeight 387
Align 380	TabIndex 387
Anchors 380	TabOrder 387
BiDiMode 381	TabPosition 388
Constraints 381	Tabs 388

TabStop 388	OnDbClick 484
TabWidth 388	OnDragDrop 484
Visible 388	OnDragOver 484
TPaletteToolList class 470	OnEndDrag 484
Align 477	OnEnter 484
AllowArrange 478	OnExit 485
Anchors 478	OnKeyDown 485
AutoCollapse 478	OnKeyPress 485
BevelEdges 478	OnKeyUp 485
BevelInner 478	OnMouseDown 486
BevelKind 479	OnMouseMove 486
BevelOuter 479	OnMouseUp 486
BiDiMode 479	OnPalChange 489
CategoryHeight 479	OnResize 486
ClsChanged 476	OnStartDrag 487
ClsPalChanged 476	ParentBiDiMode 487
Color 479	ParentColor 487
ComponentAt 476	ParentCtl3D 487
Constraints 480	ParentFont 487
Create 476	ParentShowHint 488
Ctl3D 480	PopupMenu 488
CustomItems 480	RowSpace 488
Destroy 477	ShowCaptions 488
DragCursor 481	ShowCategory 477
DragImageType 481	ShowHint 488
DragKind 481	StyleCategory 488
DragMode 481	StyleCategoryMouseOver 488
DrawItemImage 477	StyleCategorySelected 488
Enabled 481	StyleItem 488
Filtered 481	StyleItemMouseOver 488
FilterString 482	StyleItemSelected 488
FoldingIcon 482	TabOrder 489
Font 482	TabStop 489
HintProps 482	TransparentImages 489
ItemHeight 482	VerticalGroups 489
ItemIndexChanged 477	Visible 489
Items 482	TPasteInfo class 159
ItemsArranged 477	Create 159
OnCanResize 482	CurrOffset 160
OnClick 483	Destroy 160
OnConstrainedResize 483	IncForParent 160
OnContextPopup 483	Init 160

TPopupListbox class 105	Visible 65
CreateParams 106	TPropertyNameProperty class 463
CreateWnd 106	TPropertyNode class 439
ItemHeight 106	Create 441
KeyPress 106	Editor 442
OnDrawItem 106	Expandable 441
OnMeasureItem 107	GetName 441
Sorted 107	IsDefault 442
Style 107	IsReference 442
TPropertyEdit class 457	IsSubProperty 442
Component 462	Owner 441
Designer 462	PropInfo 442
ListAlign 462	ReflectModified 441
OnSetPropValueA 462	TPropertyNodes class 442
OnSetPropValueW 463	Clear 445
PropertyName 463	Create 445
ReadOnly 463	Destroy 445
TypeKinds 463	ExpandItem 445
TPropertyItem class 61	GetProps 445
Add 62	TPropListRoot class 65
Changed 62	BeginUpdate 67
Clear 62	Changed 67
Count 64	Create 67
Create 62	Destroy 68
Delete 63	EndUpdate 68
Destroy 63	ExpandItem 68
DisplayName 64	ExpCount 68
Expandable 63	ExpIndexOf 68
Expanded 64	ExpItems 68
GetName 63	Owner 69
HasValue 63	RestoreState 68
IndexOf 63	SaveState 68
Insert 63	UpdateList 68
IsEqual 63	TScaleDlg class 498
IsRoot 63	TSelFrameDlg class 499
Items 64	TSizeAdjDlg class 501
Level 64	TSmallRect class 517
Move 63	TTabOrderDlg class 502
Name 64	TTabOrderIcons class 518
Parent 65	Color 519
PathName 65	Font 519
Root 64	Height 519

Hide 519	ItemIndex 135
HorzAlign 519	Items 135
SetTabOrder 519	OnCanResize 135
Show 519	OnClick 135
VertAlign 519	OnConstrainedResize 135
Visible 519	OnContextPopup 136
Width 520	OnDbClick 136
TToolItemStyle class 123	OnDragDrop 136
Alignment 124	OnDragOver 136
BoundPen 125	OnEndDrag 137
Brush 125	OnEnter 137
Create 124	OnExit 137
Destroy 124	OnItemArranged 137
DrawItemRect 124	OnItemChanged 137
Font 125	OnMouseDown 137
OnChange 125	OnMouseMove 138
Shape 125	OnMouseUp 138
TToolList class 125	OnResize 138
Align 130	OnStartDrag 138
AllowArrange 131	ParentBiDiMode 139
Anchors 131	ParentColor 139
AutoCollapse 131	ParentCtl3D 139
BevelEdges 131	ParentFont 139
BevelInner 131	ParentShowHint 139
BevelKind 132	PopupMenu 139
BevelOuter 132	RightClickSelect 139
BiDiMode 132	RowSpace 140
CategoryHeight 132	Selected 140
Color 132	ShowHint 140
Constraints 133	StyleCategory 140
Ctl3D 133	StyleCategoryMouseOver 140
DragCursor 133	StyleCategorySelected 140
DragKind 133	StyleItem 140
DragMode 133	StyleItemMouseOver 140
Enabled 133	StyleItemSelected 140
Filtered 134	TabOrder 140
FilterString 134	TabStop 140
FoldingIcon 134	VerticalGroups 141
Font 134	Visible 141
HintProps 134	TToolListItem class 141
Images 134	Caption 142
ItemHeight 134	Expanded 142

---

Hint 142	CanMove 173
ImageIndex 142	CanPaste 173
IsCategory 142	CanRedo 170
Tag 142	CanRename 173
ToolList 143	CanResize 174
Visible 143	CanSelect 174
TToolListItems class 143	CanUndo 171
Items 143	CaptionFont 188
TUnicodeEdit class 107	CheckAction 171
Create 108	ClearCompEditorMenu 171
Destroy 108	ClearSelection 174
IsUnicode 108	ClearUndo 171
SelTextW 109	CloseDisactive 188
Text 109	CloseTextEditor 171
TextW 109	ContainerWindow 188
TzBoundCtrl class 520	CopySelection 174
Bitmap 522	Create 174
Color 522	CutSelection 174
Control 522	DeleteSelection 175
DrawMultSel 522	DesignSurface 188
GrabAtPos 521	Destroy 175
GrabSize 522	DisplayControlGrid 189
Hide 521	DisplayGrid 189
Invalidate 521	DoObjectHint 175
Local 522	DragDraw 175
Locked 522	DragDrop 171
MarkerShape 522	DragOver 171
Recreate 521	DragParentLimit 189
Update 521	Edit 175
Visible 523	EditAction 172
TzCustomFormDesigner class 161	EndDrag 176
AddCompEditorMenu 170	Events 189
AlignSelected 172	ExecuteAction 175
AlignToGrid 172	FlatIcons 189
AllowComponents 187	FlipChildren 176
AutoAlign 187	Form 189
BDSStyle 188	GetCompObj 176
BringToFront 172	GetComponent 176
BuildLocalMenu 172	GetComponentName 176
CancelDrag 173	GetComponentNames 177
CanDelete 173	GetControlAt 177
CanInsert 173	GetEditState 173

---

GetMethodName 177	OnCanRename 196
GetNewName 177	OnCanResize 196
GetObjectName 177	OnCanSelect 196
GetRoot 178	OnCreateComponent 197
GetRootClassName 178	OnCreateFrame 197
GetScriptEvent 178	OnCreateIcon 197
GetScrollRanges 178	OnCreateMethod 198
GetSelections 178	OnDrawControl 195
GetShiftState 179	OnExecuteAction 195
GridStepX 190	OnFormClosed 198
GridStepY 190	OnGetComponentHint 198
Groups 190	OnGetComponentLocked 195
GuidelinesStyle 191	OnGetMethodNames 198
IgnoreReadErrors 191	OnGetObjectName 195
Intf_Notification 179	OnGetScriptProc 199
IsComponentHidden 179	OnNotification 199
IsDesignMsg 180	OnPopUndo 196
IsLocked 179	OnPushUndo 197
IsProtected 180	OnRenameMethod 200
IsRootSelected 180	OnSetNewName 197
IsSourceReadOnly 180	OnSetScriptProc 200
KeyDown 180	OnShowMethod 201
KeyPress 181	OnUpdateAction 198
KeyUp 181	OnValidateMethod 201
LoadFromFile 177	PaintControl 184
LoadFromStream 178	PaintGrid 184
LockControls 190	PasteSelection 184
LockPublished 190	PopupMenu 191
MethodExists 181	PopupMenuFilter 192
Modified 181	ReadComp 184
MouseDown 182	ReadOnly 191
MouseMove 182	Redo 179
MouseUp 182	RenameMethod 184
MultiSelect 190	Root 192
Navigate 182	RootModified 192
NoSelection 183	SaveToFile 179
Notification 183	SaveToStream 180
NotifySelChanged 183	Scale 185
OnCanDelete 195	SelCount 193
OnCanEdit 194	SelectAll 185
OnCanInsert 195	SelectComponent 185
OnCanMove 196	Selected 193

SelectedComponent 185	GridStepY 216
SelectedComponentsCount 180	GuidelinesStyle 216
SelectionChanged 185	IgnoreReadErrors 216
SelectObj 185	LockControls 216
SelectRect 185	LockPublished 216
SelMarker 193	MultiSelect 216
SendToBack 186	OnActiveChanged 217
SetPasteName 186	OnCanDelete 217
SetScriptEvent 186	OnCanEdit 217
SetSelections 186	OnCanInsert 217
ShowCaptions 193	OnCanMove 217
ShowMethod 186	OnCanRename 218
ShowPopupMenu 186	OnCanResize 218
ShowTabOrder 182	OnCanSelect 218
SizeSelected 186	OnCreateComponent 218
SnapToGrid 193	OnCreateFrame 219
StartDrag 187	OnCreateIcon 219
StoreEvents 191	OnCreateMethod 219
TabOrderIcons 191	OnDragDrop 219
Target 193	OnDragOver 220
TextEditMode 191	OnDrawControl 220
Undo 183	OnExecuteAction 220
UndoLimit 192	OnFormClosed 220
UndoLoad 192	OnGetComponentHint 220
UniqueName 187	OnGetComponentLocked 221
UpdateAction 182	OnGetMethodNames 221
UpdateComplcons 187	OnGetObjectName 221
ValidateMethod 187	OnGetScriptProc 222
TzDesignerSelections class 445	OnHandleControlMessage 222
TzFormDesigner class 201	OnKeyDown 222
Active 214	OnKeyPress 223
AllowComponents 214	OnKeyUp 223
AutoAlign 214	OnMouseDown 223
BDSStyle 214	OnMouseMove 223
CaptionFont 214	OnMouseUp 224
CloseDisactive 214	OnNotification 224
DesignSurface 215	OnPopUndo 224
DisplayControlGrid 215	OnPushUndo 224
DisplayGrid 215	OnRenameMethod 224
DragParentLimit 215	OnSetNewName 225
FlatIcons 215	OnSetScriptProc 225
GridStepX 215	OnShowMethod 225



OnUpdateAction 225

OnValidateMethod 225

PopupMenu 226

PopupMenuFilter 226

ReadOnly 226

SelMarker 226

ShowCaptions 227

ShowHints 227

SnapToGrid 227

StoreEvents 227

TabOrderIcons 227

Target 227

TextEditMode 228

UndoLimit 229

TzMenuEditor class 515

TzMenuItemsPropertyEditor class 515

## U

Using DesignIDE.BPL 5

## V

Version 2.00 12

Version 2.10 14

Version 2.20 15

Version 2.30 17

Version 2.40 17

Version 2.50 17