

AR2Pro

---

## Table Of Contents

---

User Guide	6
Overview	6
Run-time Designer Control	6
Introduction	6
Using Run-time Designer Control	7
Adding Run-time Designer to Visual Basic	8
Adding Run-time Designer to your Project	9
Working with the Designer at Run-time	12
Saving and Loading Report Layouts	15
Using the Designer Events	18
Using Scripting	22
Custom Toolbars and Menus	25
Included Sample Projects	25
Deployment and Distribution	26
WebCache Service and ISAPI DLL	26
Introduction	26
Installation	26
Deployment	26
Using the WebCache Service	27
Developers Reference	28
ActiveReports Run-time Designer	28
ARDesigner	28
Properties	29
GridSnap	29
GridVisible	29
GridX	30
GridY	30
IsDirty	31
Locked	31
Report	32
RulerUnits	33

SelectedObjects	33
ToolbarsAccessible	34
ToolbarsVisible	34
ToolboxItem	35
Methods	36
ExecuteAction	36
GetSectionFromPoint	37
LoadFromObject	38
NewLayout	39
QueryStatus	39
SaveToObject	41
Events	41
Alert	42
ContextMenuOpen	43
Error	44
LayoutChanged	44
SelChange	45
StatusChange	45
ValidateChange	47
Selection Methods	48
Count	48
Item	48
WebCache Service	49
WebCache	49
CacheContent	49
CacheItem	50
IsCached	51
Item	52
Remove	52
RemoveAll	52
Count	53
WebCacheItem	53

Data	53
Header	54
Id	55
Persistence	55
TimeOut	56
WebCacheWorkerThread	56
TotalTimeServicingRequest	57
ThreadId	57
NumberOfRequest	58
AveragePerRequest	58
WebCacheWorkerThreads	59
Count	59
Item	60
Property List Control	60
PropList Control	60
Sorted	61
ShowToolbar	61
ShowReadOnlyProp	62
ShowObjectCombobox	62
ShowDescription	63
Properties	63
hWnd	64
ForeColor	64
Font	64
Enabled	65
Categorized	65
BorderStyle	66
BackColor	66
AllowColumnResize	66
SelectObjects	67
Refresh	67
Clear	68
AddObject	68

PropertyValidate	68
ObjectChanged	69
PropertyChanged	69
FetchDataDescription	69
FetchData	70
Error	70
ButtonClick	71
PropNode Object	71
AddEnum	71
ClearEnums	72
Category	72
Children	73
Description	73
Name	73
Type	74
Value	74
PropNodes Collection	75
Remove	75
Item	75
Count	76
Add	76

## User Guide

---

### User Guide

Overview

Run-time Designer Control

WebCache Service and ISAPI DLL

### Overview

---

#### Overview

ActiveReports Professional Edition includes three components that allow you to provide custom reporting solutions to your end users. These reporting solutions can range from a built-in customized report designer to a complete reporting and information delivery server in Internet or intranet settings.

The components include:

- | Runtime Designer Control
- | WebCache Service and ISAPI DLL
- | Property List Control

### Run-time Designer Control

---

#### Run-time Designer Control

Introduction

Using Run-time Designer Control

#### Introduction

---

#### Introduction

The run-time designer control allows you to host the ActiveReports designer your application and provide end-user report editing capabilities. The control's methods and properties provides easy access to save and load report layouts, monitor and control the design environment and customize the look and feel to the needs of your end users.

#### Persistence API

The designer control's Report property provides access to the layout elements of the report, its sections and controls. The persistence API allows you to save and load the report layout. It includes the following properties and methods.

LoadFromObject loads the report layout from an existing report object into the designer.

SaveToObject, apply the new layout to an existing report object.

NewLayout clears the current layout, including sections, controls and starts a new report layout. All property settings are returned to default values.

IsDirty, this property returns whether the report has been modified since the last save or load operation. It can be used to enable/disable a save button.

#### User Interface Customization

API's for User Interface customization have the goal of providing hooks into the designer that will let developers attach their own custom menus, toolbars, field/database browsers, script editors, alert dialogs and property sheets.

## Toolbars and Menus

You can replace built-in menus and toolbars by first setting the `ToolbarsVisible`, `ToolbarsAccessible` properties on the designer control to hide the built-in UI.

All menu and toolbar commands are called actions. There are over 50 actions that are defined in the designer control.

If you are using a pull method to update your toolbar and menu states using idle-time processing, you can use the `QueryStatus` method to check if a certain action (such as `Edit/Cut`) is enabled/disabled, checked/unchecked.

In addition, the designer control fires `StatusChange` event when the status of the tools change allowing you to update the UI to reflect these changes.

`ExecuteAction` method provides the ability to perform most of the designer functions with a single call. Alternatively, actions that are not supported by `ExecuteAction` (ones that require a parameter such as color, style and font settings) can be executed by setting the control or section properties directly using the `Report` property.

## Designer Surface

The grid settings can be modified using the following properties

- | `GridX` and `GridY` determine the number of grid points in each ruler unit.
- | `GridVisible` determines whether the grid is visible or not.
- | `GridSnap` specifies whether the controls should snap to the visible grid points.
- | `RulerUnits` allows you to select ruler units from either US or metric units.

## Property Sheets

The runtime designer control allows you to replace the built-in property toolbox and provide your own selection editing UI. The `SelChange` event fires when the user changes the current selected object in the designer. You can retrieve a list of the selected object using the `SelectedObjects` collection.

ActiveReports Professional includes a property listbox `ActiveX` called "Data Dynamics Property ListBox" that can be used to create customized design environments based on your users needs.

## Script Editor

The built-in syntax-highlighting script editor is invoked using the `ExecuteAction` method and the action code `ddActionCodeEditor`. To replace it with your own editor, create your own toolbar/menu item and use the `ActiveReport.Script`, `Section.Script` properties to get/set the script. The scripting language is can be set using the `ActiveReport.ScriptLanguage` property.

## Controls Toolbox

The toolbox contains the controls that can be placed on a section. You can create your own toolbox toolbar and use the following properties and methods to interface with the designer:

**ToolboxItem property:** Setting the `ToolboxItem` property initiates the control-add mode using the `ProgID` set to the property. The user will use the rubber-band to select the area of the control and once the area is selected the designer will add the control specified by `ProgID` and end the add mode by setting `ToolboxItem` to an empty string.

**ValidateChange Event:** This event fires after any changes that are made to the report layout. It allows you to control what the user can or cannot do in the designer control. Within the event code you can cancel the layout change and revert it back to it's original state.

**LayoutChanged Event:** After the layout change (control addition, deletion is validated this event will fire with `changeType=ddLCCControlAdd` to notify the application that a new control has been added.

## Alerts and Error Messages

ActiveReports runtime designer allows you to intercept runtime errors and alert messages and present the user with custom notification UI. For each error or alert message `ActiveReport Designer` control fires an `Error` or `Alert` event with the message id and string and gives you the option to cancel the internal display when you handle the messages.

## Using Run-time Designer Control

## Using the Run-time Designer Control

Adding Run-time Designer to Visual Basic

Adding Run-time Designer to your Project

Working with the Designer at Run time

Saving and Loading Report Layouts

Using the Designer Events

Using Scripting

Custom Toolbars and Menus

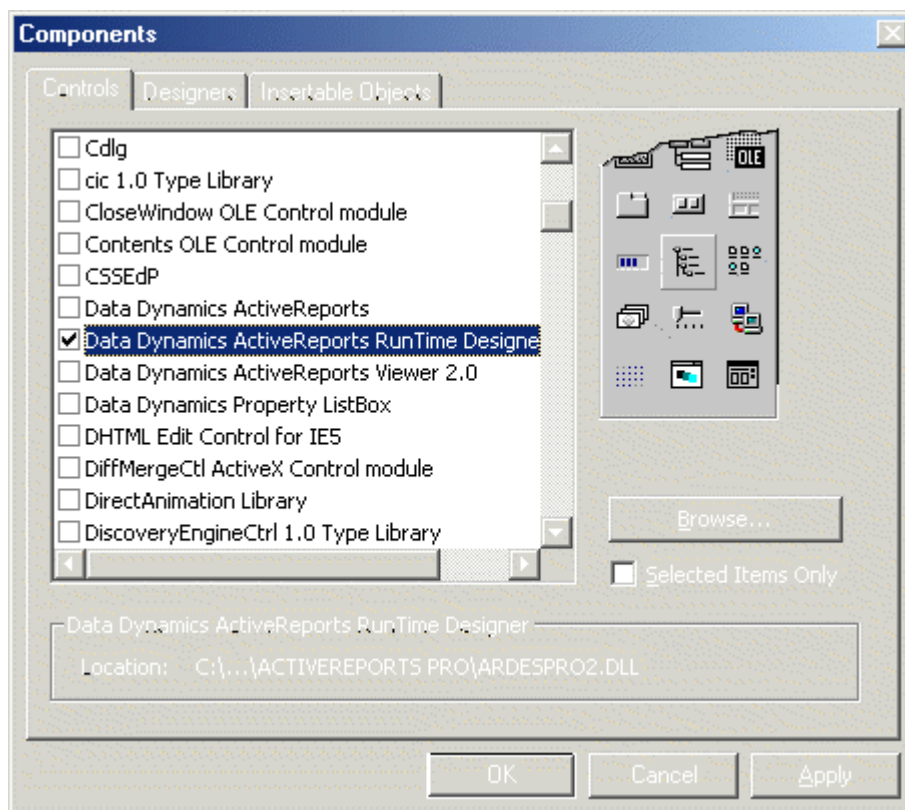
Deployment and Distribution

## Adding Run-time Designer to Visual Basic

### Adding Run-time Designer to Visual Basic

The end-user designer is an ActiveX control; the following steps describe how to include it in the Visual Basic IDE:

1. Start Visual Basic.
2. Choose Project > Components (Ctrl-T).



3. Choose Data Dynamics ActiveReports Run-time Designer.

Note: If the run-time designer entry does not appear in the list, make sure that Selected Items Only is not checked. If it still does not appear, make sure ARdespro2.dll is registered by running regsvr32 on ARdespro2.dll.

4. Click OK to close the dialog box.




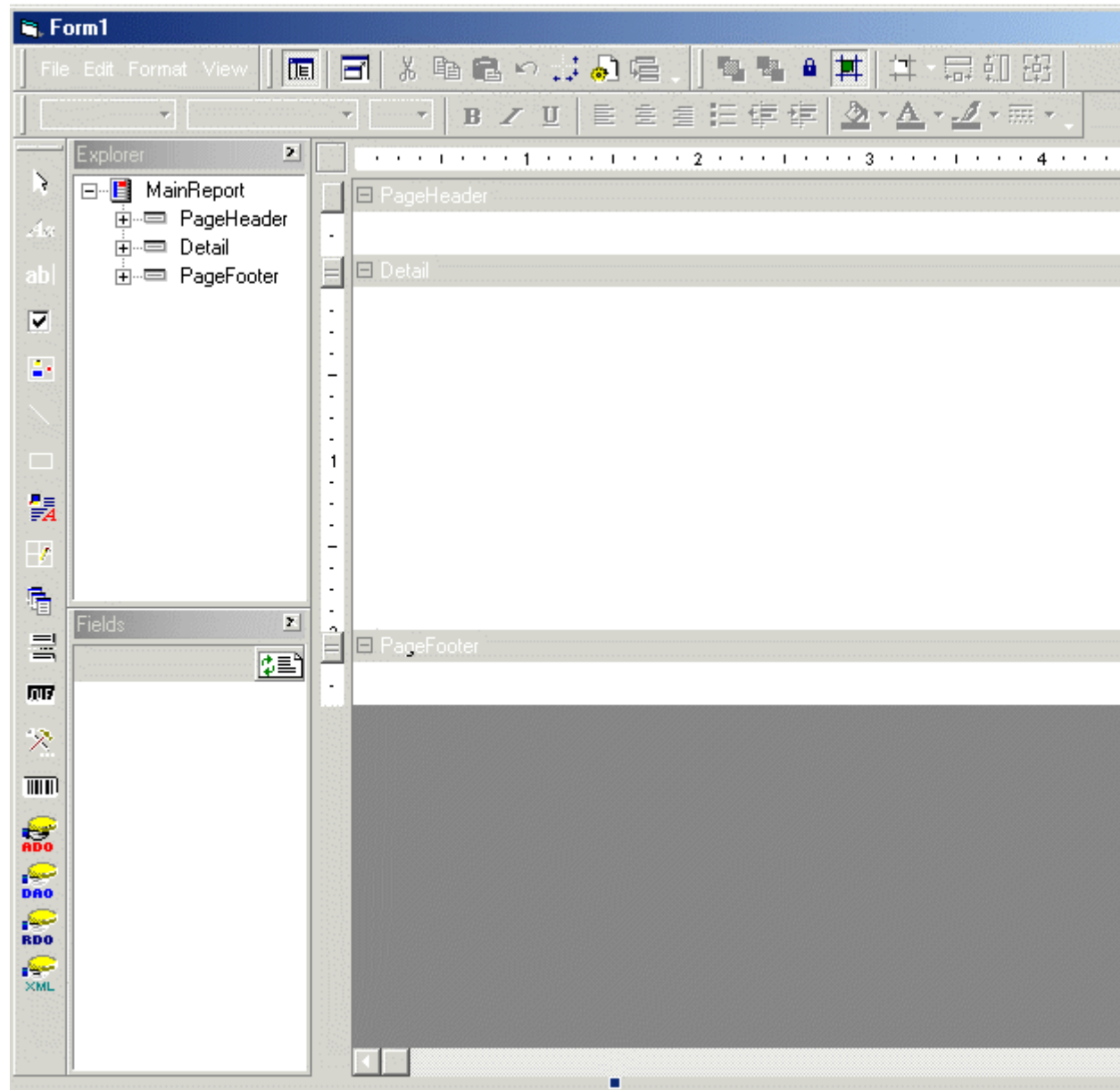
5. The run-time designer icon  should appear in the toolbox.

## Adding Run-time Designer to your Project

---

### Adding Run-time Designer to your Project

1. Click on the run-time designer icon  in the toolbox.
2. Place the control on the form (shown below) and size it accordingly.



The run-time designer's appearance is the same as the ActiveReports ActiveX designer but the end user will not have direct access to the reporting events in Visual Basic. Instead, the user will use VBScript or JScript to handle the reporting events as needed. The run-time designer includes a syntax-highlighting editor for both languages.

The following sample demonstrates adding the run-time designer to a Visual Basic project and using ActiveReport's viewer control to view reports designed at run time.

1. Start a new Visual Basic standard EXE project.

2. Select the following components from Visual Basic's components list:
  - i Data Dynamics ActiveReports Runtime Designer
  - i Data Dynamics ActiveReports Viewer 2.0
  - i Microsoft Tabbed Dialog Control
3. Add the following references from Visual Basic's reference list:
  - i Data Dynamics ActiveReports 2.0
4. Select Form1 and set its properties as follows:

Name	frmMain
Caption	Simple Designer Project
Height	9465
Width	11295

5. Add a SSTab control to frmMain and set its properties as follows:

Height	9015
Left	0
Tabs	2
Top	0
Width	11175

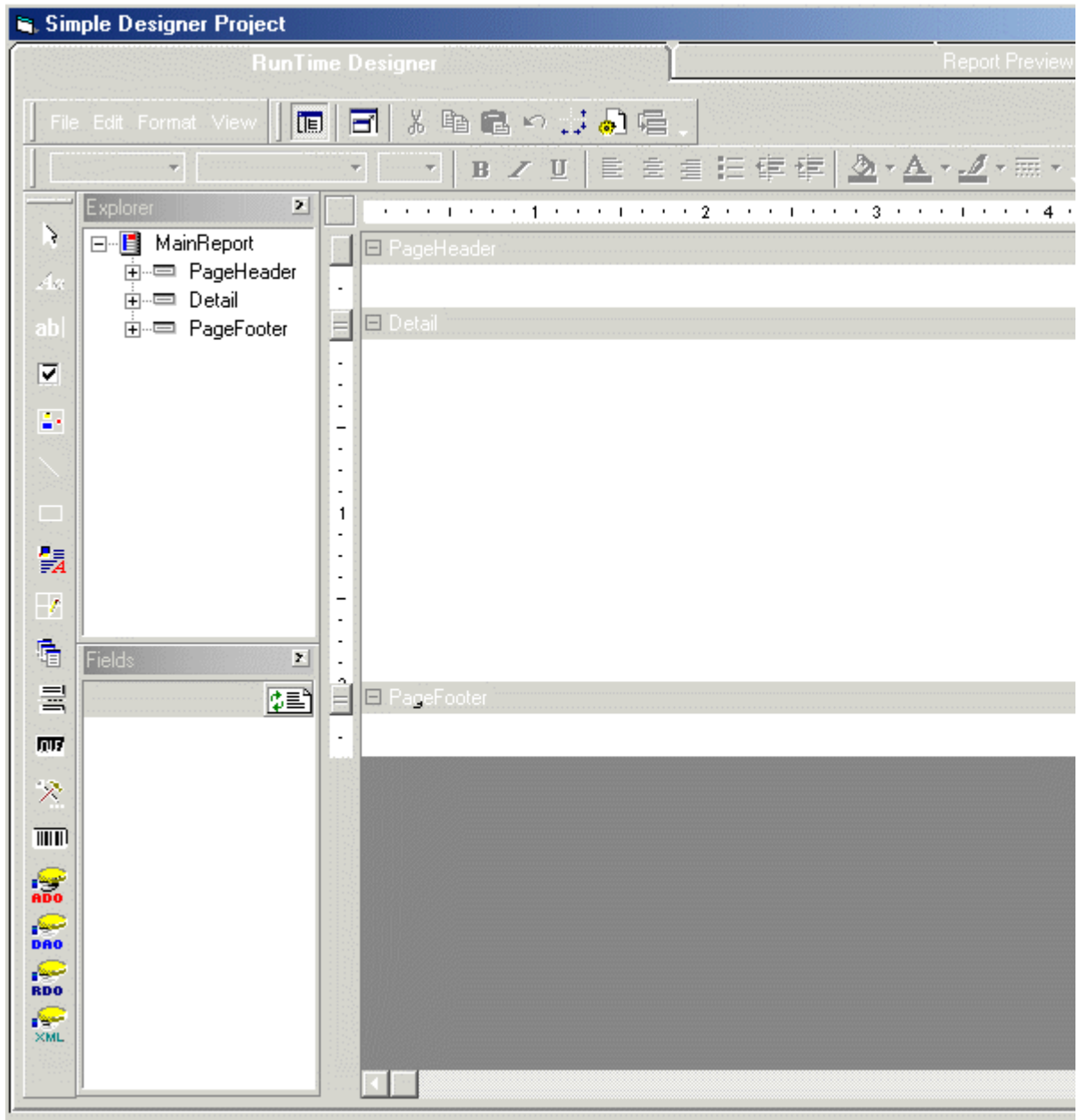
6. Right-click on SSTab1 and select properties.
7. Set the TabCaption for Tab0 to Run-time Designer.
8. Set the TabCaption for Tab1 to Report Preview and select OK to close the tab control's property page.
9. Add the run-time designer to Tab0 and set its properties as follows:

Name	ard
Height	8415
Left	120
Top	480
Width	10935

10. Add the viewer control to Tab1 and set its properties as follows:

Name	arv
Height	8535
Left	120
Top	360
Width	10935

11. frmMain should look like this:



12. Add the following code to the Form\_Load event:

```
Dim rpt As DDAciveReports2.ActiveReport

Private Sub Form_Load()
    'Set active Tab to the designer
    SSTab1.Tab = 0
    Set rpt = New ActiveReport
    'Activate all the toolbars
    ard.ToolbarsVisible = ddTBToolBox + ddTBAlignment + ddTBExplorer + _
        ddTBFields + ddTBFormat + ddTBMenu + ddTBPropertyToolbox + ddTBStandard
    ard.ToolbarsAccessible = ddTBToolBox + ddTBAlignment + ddTBExplorer + _
        ddTBFields + ddTBFormat + ddTBMenu + ddTBPropertyToolbox + ddTBStandard
End Sub
```

Note: When working with the designer, the toolbars cannot be customizing. The only available options

are ToolbarsVisible and ToolbarsAccessible. If the project requires custom toolbars, a third party toolbar control will need to be substituted for the runtime designer's toolbars.

13. Add the following code to the SSTab1\_Click event:

```
Private Sub SSTab1_Click(PreviousTab As Integer)
    Select Case PreviousTab
        Case Is = 0
            prepPreview
        Case Is = 1
            prepDesigner
    End Select
End Sub
```

14. Add the following code to prepare the viewer control and designer when its tab is selected:

```
Private Sub prepPreview()
    On Error GoTo errHndl
    'Must be used to writes the designer's layout
    'to the report so it can be previewed
    ard.SaveToObject rpt
    rpt.Restart
    'Run the new report
    rpt.Run False
    'Add the report to the veiwer
    Set arv.ReportSource = rpt
    Exit Sub

errHndl:
    MsgBox "Error Previewing the Report: " & Err.Number & " " & Err.Description
End Sub

Private Sub prepDesigner()
    On Error GoTo errHndl

    If Not arv.ReportSource Is Nothing Then
        arv.ReportSource.Cancel
        Set arv.ReportSource = Nothing
    End If

    Exit Sub
errHndl:
    MsgBox "Error in Design Preview: " & Err.Number & " " & Err.Description
End Sub
```

Note: SaveToObject must be used to save the changes made in the run-time designer to an ActiveReport report object. You should always use that object to run and preview the report, do NOT use the designer's Report property to run and preview the report.

15. Save and run the project.
16. While the project is running, continue on to the next sample for a demonstration on using the designer at run time.

## Working with the Designer at Run-time

---

### Working with the Designer at Run time

This sample demonstrates the fundamentals of using the designer at run time. The simple report created in this sample will be used to demonstrate more advanced features later on in the manual. At runtime the designer functions similarly to the ActiveX designer but does not allow access to the report events or code.

1. Start by running the sample project created above.
2. Place an ADO data control in the designer's detail section.

3. Connect to Nwind.mdb (see chapter 3 in the standard edition user's guide).  
Note: The samples in this manual use the NorthWind database included with Microsoft Visual Basic.
4. Set the DataControl's source property to the following SQL statement:  
SELECT \* FROM customers order by country
5. Right-click on the designer and select insert to add a new GroupHeader/Footer.
6. Click on the new section GroupHeader1 to select it.
7. Modify the section's properties as follows:
 

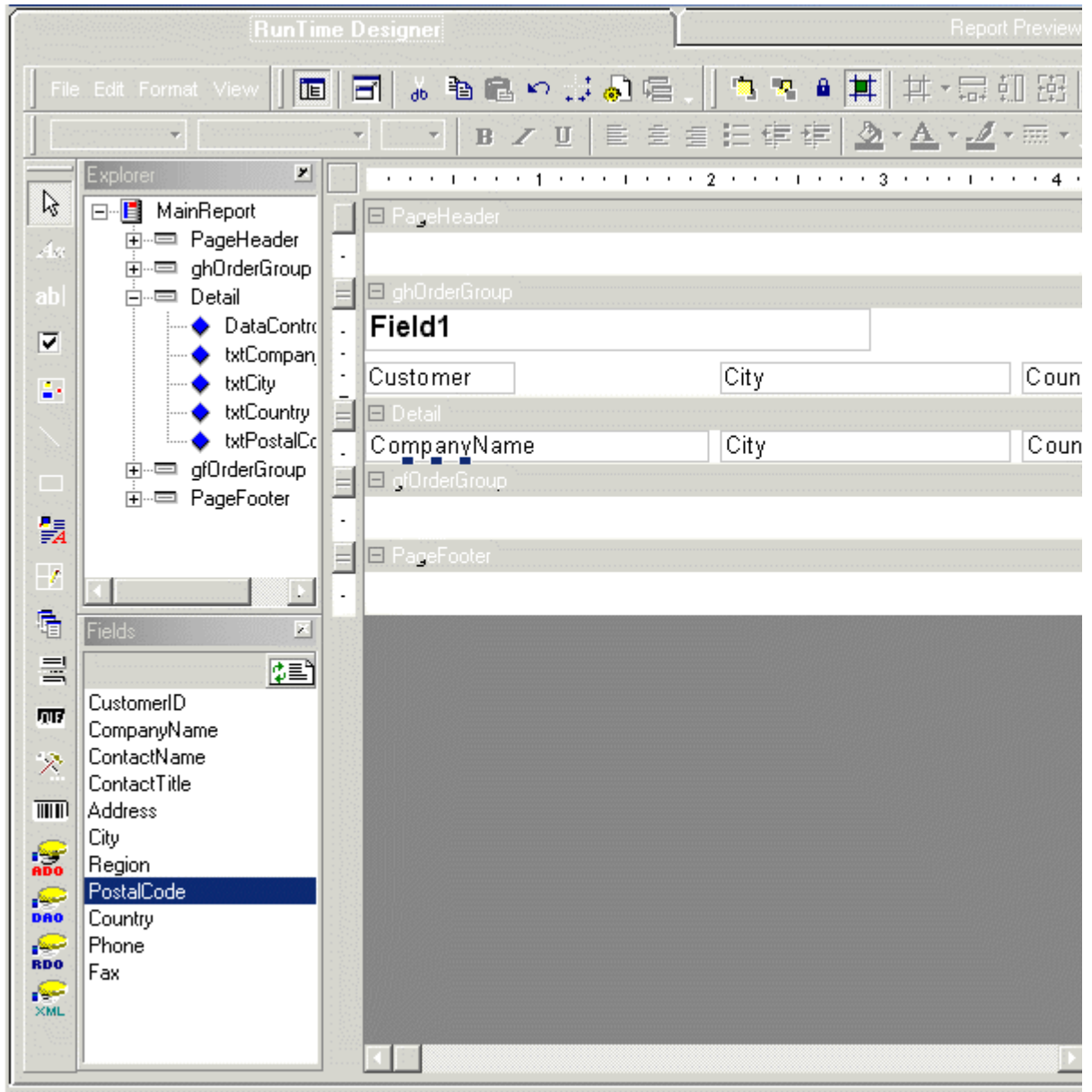
Name	ghOrderGroup
DataField	Country
Height	750
8. Click on the new section GroupFooter1 to select it.
9. Modify the section's properties as follows:
 

Name	gfOrderGroup
Height	270
10. Add a Field control to the ghOrderGroup section and set its properties as follows:
 

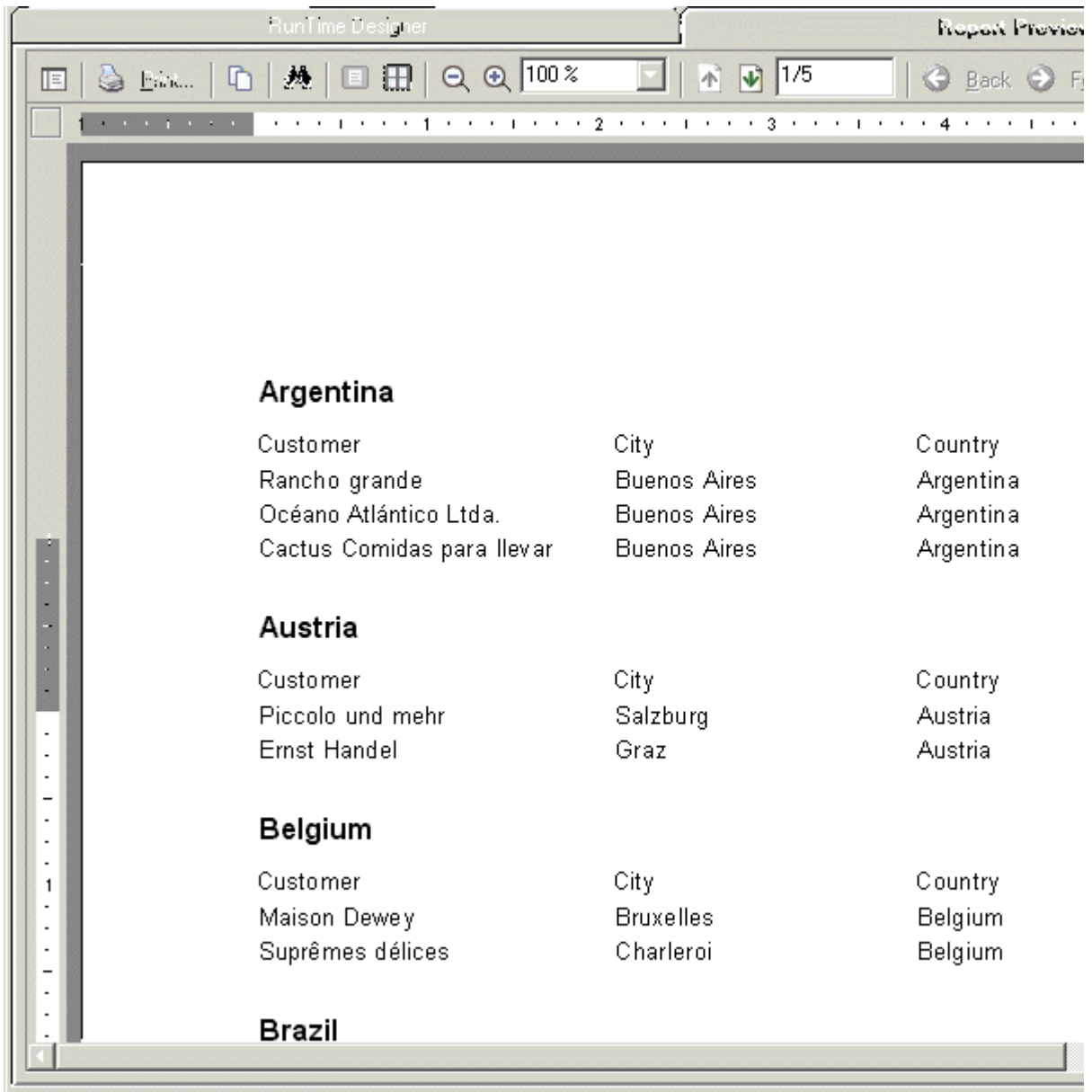
Name	txtGroupCountry
DataField	Country
Height	360
Left	0
Top	0
Width	4230
Font .Size	12
Font .Bold	True
11. Place 4 labels in the ghOrderGroup section and set their properties as follows:
 

Name	lblCustomer	lblCity	lblCountry	lblPostalCode
Caption	Customer	City	Country	PostalCode
Height	270	270	270	270
Left	0	2970	5490	7380
Top	450	450	450	450
Width	2880	2430	1800	1800
12. Click and drag the following fields from the fields list into the detail section: CompanyName, City, Country and PostalCode.
13. Set the field's properties as follows:
 

Name	txtCustomer	txtCity	txtCountry	txtPostalCode
DataField	CompanyName	City	Country	PostalCode
Height	270	270	270	270
Left	0	2970	5490	7380
Top	0	0	0	0
Width	2880	2430	1800	1800
Alignment	0-Left	0-Left	0-Left	1-Right
14. Set the detail sections height to 285.
15. The designer should look like this:



- Click on the Report Preview tab to run and show the report.



17. Switch back to the Runtime Designer tab and follow the next sample to see how the designer's layout can be saved.

## Saving and Loading Report Layouts

### Saving and Loading Report Layouts

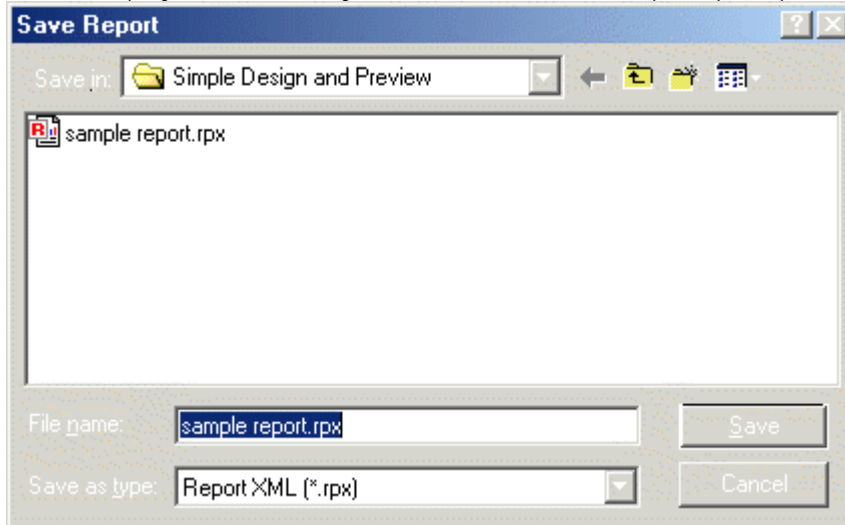
Reports can be saved and loaded into the designer by a variety of different methods. The easiest method is to use the File menu on the designer to Save or Open RPX files (ActiveReport's standard XML-formatted report files).

#### Open/Save From File Menu

To save the report created in the previous sample:

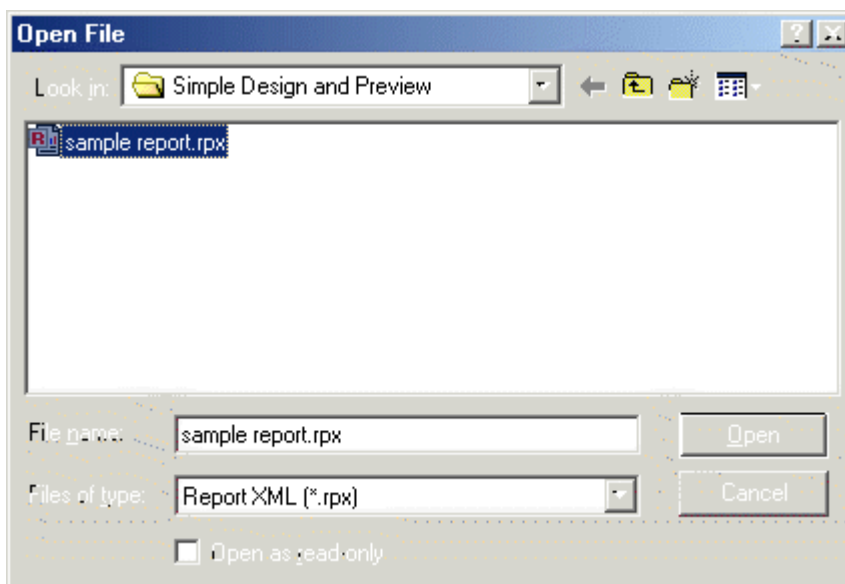
1. Select the File menu.
2. Select the Save menu option.

3. Select the project's directory, set the File name to sample report.rpx and select save.



Stop the project and restart is so the designer will return to the default setting. To load the previously created report back into the designer:

1. Select the File menu.
2. Select the Open menu option.
3. Select the sample report.rpx file from the project's directory and select Open.



When the RPX file is loaded, the designer will display the previously created report.

### Open/Save Through Code

A designer's layout can be saved and loaded through code by using the following methods:

Saving:

To save a designer layout in code use the designer's SaveToObject method to save the layout to a report object. Once the layout is saved to the report object, the report object's SaveLayout method can be used to save the layout to an RPX file, or byte array. Add the following code to the sample project to save the designer layout whenever the Report Preview tab is selected.

```
Private Sub prepPreview()  
    On Error GoTo errHndl  
    'Writes the designer's layout
```



```

'to the report so it can be previewed.
ard.SaveToObject rpt
'Saves the report object to the specified style
rpt.SaveLayout App.Path & "\sample report.rpx", ddSOFile
'Resets report
rpt.Restart
'Run the new report
rpt.Run False
'Add the report to the veiwier
Set arv.ReportSource = rpt
Exit Sub

errHndl:
MsgBox "Error Previewing the Report: " & Err.Number & " " & Err.Description
End Sub

```

Save these changes.

Loading:

To load a designer layout in code use the report object's Load method to load a specified RPX file and the designer's LoadFromObject to read the layout into the designer. Add the following code to the project to load the report designer when the project starts, and whenever the Runtime Designer tab is selected.

```

Private Sub Form_Load()
'Set active Tab to the designer
SSTab1.Tab = 0
Set rpt = New ActiveReport
'Activate all the toolbars
ard.ToolbarsVisible = ddTBToolBox + ddTBAlignment + ddTBExplorer + _
    ddTBFields + ddTBFormat + ddTBMenu + ddTBPropertyToolbox + ddTBStandard

ard.ToolbarsAccessible = ddTBToolBox + ddTBAlignment + ddTBExplorer + _
    ddTBFields + ddTBFormat + ddTBMenu + ddTBPropertyToolbox + ddTBStandard
'Load the saved RPX file into a report object
rpt.LoadLayout App.Path & "\sample report.rpx"
'Load the report object into the designer
ard.LoadFromObject rpt
End Sub

Private Sub prepDesigner()
On Error GoTo errHndl

If Not arv.ReportSource Is Nothing Then
    arv.ReportSource.Cancel
    Set arv.ReportSource = Nothing
End If

'Load the saved RPX file into a report object
rpt.LoadLayout App.Path & "\sample report.rpx"
'Load the report object into the designer
ard.LoadFromObject rpt

Exit Sub
errHndl:
MsgBox "Error in Design Preview: " & Err.Number & " " & Err.Description
End Sub

```

Save these changes.

Loading DSR (ActiveX Designer) Files

The run-time designer can also load ActiveReport's ActiveX Designers included within the project. To demonstrate this capability:

1. Add an ActiveReport ActiveX Designer to the project and set its properties as follows.
  - Name rptSample
2. From the designer's File menu, open the previously saved sample report.rpx file. When the RPX file is

opened the ActiveX designer will have the same report that was developed with the runtime designer.

3. Modify frmMain's Form\_Load event to load rptSample instead by adding the following code:

```
Private Sub Form_Load()
    'Set active Tab to the designer
    SSTab1.Tab = 0
    Set rpt = New ActiveReport
    'Activate all the toolbars
    ard.ToolbarsVisible = ddTBToolBox + ddTBAlignment + ddTBExplorer + _
        ddTBFields + ddTBFormat + ddTBMenu + ddTBPropertyToolbox + ddTBStandard
    ard.ToolbarsAccessible = ddTBToolBox + ddTBAlignment + ddTBExplorer + _
        ddTBFields + ddTBFormat + ddTBMenu + ddTBPropertyToolbox + ddTBStandard
    'Load the ActiveX designer into the run-time designer
    ard.LoadFromObject rptSample
End Sub
```

Do not save these changes.

## Using the Designer Events

---

### Using the Designer Events

The runtime designer uses four main events to control the actions performed by the end user. These events are LayoutChanged, SelChange, StatusChange and ValidateChange.

#### LayoutChanged

LayoutChanged fires when the designer's layout is changed. The event can be used to monitor changes made to the report layout and update any dependent data such as SQL queries or custom user interfaces. The following list gives a description for the different layout changes.

Setting	Description
ddLCControlMove	0 • A control's position has changed.
ddLCControlSize	1 • A control's size has changed.
ddLCControlDelete	2 • A control has been deleted.
ddLCSectionSize	3 • A section's size has changed.
ddLCSectionDelete	4 • A section is deleted.
ddLCSectionMove	5 • A section is moved.
ddLCReportSize	6 • The report's size is changed.
ddLCControlAdd	7 • A new control has been added to the report.

#### SelChange

SelChange fires when an item in the designer is selected. The event can be used to identify the selected item by accessing the designer's SelectedObjects property.

#### StatusChange

StatusChange fires for each change in the status of the designer action. Designer actions represent the commands typically invoked from UI elements such as toolbars or menus. The following list gives a description for all of the actions:

Setting	Description
ddActionFOpen	1 - File: Open.
ddActionFSave	2 - File: Save.
ddActionFPageSetup	3 - File: Page Setup.
ddActionECut	4 - Edit: Cut.

ddActionEPaste	5 - Edit: Paste.
ddActionECopy	6 - Edit: Copy.
ddActionEUndo	7 - Edit: Undo.
ddActionEDelete	8 - Edit: Delete.
ddActionEDeleteSection	9 - Edit: Delete Section.
ddActionEInsertReportHF	10 - Edit: Insert Report Header/Footer.
ddActionEInsertPageHF	11 - Edit: Insert Page Header/Footer.
ddActionEInsertGroupHF	12 - Edit: Insert Group Header/Footer.
ddActionEReorderGroups	13 - Edit: Reorder Groups.
ddActionEInsertField	14 - Edit: Insert Field.
ddActionViewExplorer	15 - View: Report Explorer.
ddActionViewFieldsList	16 - View: Fields List.
ddActionViewPropertyList	17 - View: Property Listbox.
ddActionViewGrid	18 - View: Grid.
ddActionViewSnapToGrid	19 - View: Snap to grid.
ddActionViewFullScreen	20 - View: Full screen.
ddActionViewCodeEditor	21 - View: Script Code Editor.
ddActionFoAlignLefts	22 - Format: Align Control Lefts.
ddActionFoAlignRights	23 - Format: Align Control Rights.
ddActionFoAlignCenters	24 - Format: Align Control Centers.
ddActionFoAlignTops	25 - Format: Align Control Tops.
ddActionFoAlignMiddles	26 - Format: Align Control Middles.
ddActionFoAlignBottoms	27 - Format: Align Control Bottoms.
ddActionFoAlignToGrid	28 - Format: Align to Controls Grid.
ddActionFoAlignCenterInSec	29 - Format: Align: Center Control in Section.
ddActionFoSizeSameWidth	30 - Format: Size controls to the same width.
ddActionFoSizeSameHeight	31 - Format: Size controls to the same height.
ddActionFoSizeSameBoth	32 - Format: Size controls to the same width and height.
ddActionFoVSpaceEqual	33 - Format: Space controls even vertically.
ddActionFoVSpaceIncrease	34 - Format: Increase vertical spacing.
ddActionFoVSpaceDecrease	35 - Format: Decrease vertical spacing.
ddActionFoHSpaceEqual	36 - Format: Space controls even horizontally.
ddActionFoHSpaceIncrease	37 - Format: Increase horizontal spacing.
ddActionFoHSpaceDecrease	38 - Format: Decrease horizontal spacing.
ddActionFoOrderBringToFront	39 - Format: Bring control to the foreground.
ddActionFoOrderSendToBack	40 - Format: Send control to the background.
ddActionFoLockControls	41 - Format: Lock controls size and position.
ddActionFoStyle	42 - Format: Style.
ddActionFoFontName	43 - Format: Font name.
ddActionFoFontSize	44 - Format: Font size.
ddActionFoFontBold	45 - Format: bold.
ddActionFoFontItalic	46 - Format: Italic.
ddActionFoTextAlignLeft	47 - Format: Align text left.
ddActionFoTextAlignCenter	48 - Format: Align text center.
ddActionFoTextAlignRight	49 - Format: Align text Right.
ddActionFoForeColor	50 - Format: Set foreground color.
ddActionFoBackColor	51 - Format: Set background color.
ddActionFoLineStyle	52 - Format: Set line style.
ddActionFoLineColor	53 - Format: Set line color.
ddActionFoBorder	54 - Format: Set border styles.
ddActionFoBullets	55 - Format: Set bullet style.
ddActionFoIndent	56 - Format: Indent text.
ddActionFoOutdent	57 - Format: Outdent text.
ddActionFoUnderline	58 - Underline.

Note: The ExecuteAction method can be used to execute most of the actions above. The items that cannot

be executed with this method are items requiring parameters, such as color, font, size and style.

#### ValidateChange

ValidateChange fires before an item is moved, sized or deleted. This event can be used to control the end user's actions. For instance, this event can be used to prevent the user from removing or moving an important control.

These events can be demonstrated by adding the following to the sample project.

1. Select the following components from Visual Basic's components list:

Microsoft Windows Common Controls 6.0

Microsoft Common Dialog Control 6.0

2. Add a status bar to the bottom of frmMain and change its name to sb.
3. Add a second panel to the status bar and set its AutoSize property to 1-sbrSpring.
4. Add a common dialog control to frmMain and set its name to cmDLG.
5. Add the following main menu item to Visual Basic's menu editor:

Caption	&File
Name	mFile

6. Add the following submenu item to the File menu:

Caption	&Exit
Name	mExit

7. Add the following second main menu item to the menu editor:

Caption	&Edit
Name	mEdit

8. Add the following submenu item to the Edit menu:

Caption	&Font
Name	mFont

9. Modify the projects code to handle the added menu items:

```
Private Sub mExit_Click()
    Unload Me
End Sub

Private Sub mFont_Click()
    'Show the font dialog box
    cmDLG.Flags = cdlCFBoth
    cmDLG.ShowFont

    'Updated the selected item(s) with the new font specs
    For x = 0 To ard.SelectedObjects.Count - 1
        ard.SelectedObjects(x).Font.Name = cmDLG.FontName
        ard.SelectedObjects(x).Font.Size = cmDLG.FontSize
        ard.SelectedObjects(x).Font.Underline = cmDLG.FontUnderline
        ard.SelectedObjects(x).Font.Italic = cmDLG.FontItalic
    Next x
End Sub
```

10. Modify the prepPreview and prepDeisgner subs to handle the menu items:

```
Private Sub prepPreview()
    On Error GoTo errHndl
    'Writes the designer's layout
    'to the report so it can be previewed.
    ard.SaveToObject rpt
    'Saves the report object to the specified style
    rpt.Save App.Path & "\sample report.rpx", ddSOFile
    'Resets report
```

```

rpt.Restart
'Run the new report
rpt.Run False
'Add the report to the veiw
Set arv.ReportSource = rpt

'Disable menu items in preview mode
mFile.Enabled = False
mEdit.Enabled = False

Exit Sub

errHndl:
MsgBox "Error Previewing the Report: " & Err.Number & " " & Err.Description
End Sub

Private Sub prepDesigner()
    On Error GoTo errHndl

    If Not arv.ReportSource Is Nothing Then
        arv.ReportSource.Cancel
        Set arv.ReportSource = Nothing
    End If

    'Load the saved RPX file into a report object
    rpt.Load App.Path & "\sample report.rpx"
    'Load the report object into the designer
    ard.LoadFromObject rpt

    'Enable the menu items in design mode
    mFile.Enabled = True
    mEdit.Enabled = True

    Exit Sub
errHndl:
MsgBox "Error in Design Preview: " & Err.Number & " " & Err.Description
End Sub

```

11. Add the following code to the project to handle each of the above events:

```

Private Sub ard_LayoutChanged(ByVal changedObject As Object, ByVal changeType As DDActiveI
    Dim cnv As DDActiveReports2.Canvas
    Dim w As Long, h As Long
    Dim sLCaption As String

    'The following code checks to see if a lable has been added
    'If a label is added, it will prompt the user for a caption
    'And set the lable's width and height to fit the caption

    'Check if a label as been added
    If TypeOf changedObject Is DDActiveReports2.Label And changeType = ddLCControlAd
        'Get a caption for the label
        sLCaption = InputBox("Enter a Caption for the Label", "Enter Caption")

    'If no caption is given, use the added object's name
    If sLCaption = "" Then sLCaption = changedObject.Name

        'Set the added label's caption to the given caption
        changedObject.Caption = sLCaption

        'Use the canvas object to get a width and height for the caption
        Set cnv = New DDActiveReports2.Canvas

        'makes sure the canvas is measures with the same font size
        cnv.Font = changedObject.Font
        cnv.MeasureText sLCaption, w, h

```

```

        'Change the added controls width and height
        changedObject.Width = w
        changedObject.Height = h

        'unload the canvas
        Set cnv = Nothing
    End If

End Sub

Private Sub ard_SelChange()
    Dim sControl As String
    'Following code displays the selected label or field's name,
    'Top, left, height and width
    If ard.SelectedObjects.Count = 1 Then
        If TypeOf ard.SelectedObjects(X) Is DDActiveReports2.Field Or _
            TypeOf ard.SelectedObjects(X) Is DDActiveReports2.Label Then
            sControl = ard.SelectedObjects(X).Name
            sControl = sControl & " Top:" & ard.SelectedObjects(X).Top
            sControl = sControl & " Left:" & ard.SelectedObjects(X).Left
            sControl = sControl & " " & ard.SelectedObjects(X).Height & _
                " twips X "
            sControl = sControl & ard.SelectedObjects(X).Width & " twips"
        End If
    Else
        sControl = ""
    End If
    sb.Panels(2).Text = sControl
End Sub

Private Sub ard_StatusChange(ByVal action As DDActiveReportsDesignerCtl.DesignerActionType)
    Select Case action
        Case ddActionFoFontName
            'Enable/Disable the font menu option
            mFont.Enabled = ard.QueryStatus(ddActionFoFontName)
    End Select
End Sub

Private Sub ard_ValidateChange(ByVal changedObject As Object, ByVal changeType As DDActiveReportsDesignerCtl.DesignerActionType)
    'The following code prevents the end user from deleting the
    'Data control
    If TypeName(changedObject) = "DataControl" Then
        If changeType = ddLCControlDelete Then
            MsgBox "You are not allowed to delete the report's data control" & vbCrLf & "Cannot Remove Control"
            Cancel = True
        End If
    End If
End Sub

```

12. Save and run the project.

## Using Scripting

---

### Using Scripting

When working with RPX files, all necessary report code must be included with the RPX file in the form of a script because any Visual Basic code used to create the report is not saved into the RPX file. Also, the end user will need to use an ActiveScripting language to make any type of programmatic changes to a report.

Note: For a more detailed explanation of scripting examine chapter 14 in the standard edition user's guide.

ActiveReports provides two different methods to help make scripting easier and more versatile with Visual Basic. The report object's AddCode method allows code to be added, in the form of a string, at runtime and the AddNamedItem method adds functions and subs contained inside the Visual Basic code to the scripting name space. Continuing with the designer sample we will use both methods to demonstrate how each item is setup. Because RPX files are not secure files, it is highly suggested that all sensitive information be left out of

the RPX file. Since the project is currently using a data control, with the connection string specified, the connection string will be visible in the RPX file. It is highly recommended to use AddNamedItem to allow the Visual Basic project to retrieve the Recordset and pass this to the DataControl. The following demonstrates how to convert the sample project to take advantage of the AddNamedItem method.

Using AddNamedItem

1. Add a class module to the project and set its name to clsFunctions.

Note: When working with AddNamedItem, the subs and functions must be wrapped within a class.

2. In Visual Basic's references list, select the newest Microsoft ActiveX Data Objects Library.
3. Add the following function to clsFunctions:

```
Public Function getRSet() As ADODB.Recordset
    Dim rs As ADODB.Recordset
    Dim cn As ADODB.Connection
    Dim cnnString As String
    On Error GoTo errHndl

    Set cn = New ADODB.Connection
    Set rs = New ADODB.Recordset

    'Connect to DB and get recordset
    cnnString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program Files\Micro
cn.Open cnnString
rs.Open "Select * from customers order by country", cn
Set getRSet = rs

    Set rs = Nothing
    Set cn = Nothing

    Exit Function

errHndl:
    MsgBox "Unable to get recordset: " & Err.Number & ": " & Err.Description
    Set rs = Nothing
    Set cn = Nothing
End Function
```

4. Make the following modifications to the prepViewer sub to make the report object and script aware of the added class:

```
Private Sub prepPreview()
    On Error GoTo errHndl
    'Writes the designer's layout
    'to the report so it can be previewed.
    ard.SaveToObject rpt
    'Saves the report object to the specified style
    rpt.Save App.Path & "\sample report.rpx", ddSOFile
    'Resets report

    'Activate the Script debugger and refresh the script
    rpt.ScriptDebuggerEnabled = True
    rpt.ResetScripts

    ' Use AddNamedItem to add the function to the scripting name space
    rpt.AddNamedItem "vbCode", New clsFunctions

    rpt.Restart
    'Run the new report
    rpt.Run False
    'Add the report to the viewer
    Set arv.ReportSource = rpt


    'Disable menu items in preview mode
    mFile.Enabled = False
    mEdit.Enabled = False
```

```

Exit Sub

errHndl:
MsgBox "Error Previewing the Report: " & Err.Number & " " & Err.Description
End Sub

```

5. Save and run the project.
6. Select DataControl1 on the designer and clear out the ConnectionString and Source string.
7. Select the Script icon  and add the following code to the ActiveReport Document OnDataInitialize sub:

```

Sub OnDataInitialize
    set rpt.datacontrol1.recordset = vbcode.getrset
End Sub

```

8. Select the Report Preview tab to use the new function.

Using AddCode

1. Add the following code to clsFunctions:

```

Public Function IIf(Expression, TruePart, FalsePart)
    IIf = VBA.IIf(Expression, TruePart, FalsePart)
End Function

Public Function Format(Expression, sFormat)
    Format = VBA.Format(Expression, sFormat)
End Function

```

2. Add the following code to frmMain:

```

Private Function HelperCode() As String
    Dim sCode As String
    sCode = ""
    sCode = sCode & _
        "Public Function IIf(expr, exprTrue, exprFalse)" & vbCrLf & _
        "If expr Then IIf = exprTrue Else IIf = exprFalse" & vbCrLf & _
        "End Function"

    sCode = sCode & _
        "Public Function Format(expr, fmt)" & vbCrLf & _
        "Format = vbCode.Format(expr, fmt)" & vbCrLf & _
        "End Function"
End Function

```

3. Add the following code to prepPreview to use the AddCode method:

```

Private Sub prepPreview()
    On Error GoTo errHndl
    'Writes the designer's layout
    'to the report so it can be previewed.
    ard.SaveToObject rpt
    'Saves the report object to the specified style
    rpt.Save App.Path & "\sample report.rpx", ddSOFile
    'Resets report

    'Activate the Script debugger and refresh the script
    rpt.ScriptDebuggerEnabled = True
    rpt.ResetScripts

    'Add IIf helper code
    rpt.AddCode HelperCode()

    'Use AddNamedItem to add the function to the scripting name space

```



```

rpt.AddNamedItem "vbCode", New clsFunctions

rpt.Restart
'Run the new report
rpt.Run False
'Add the report to the veiwier
Set arv.ReportSource = rpt

'Disable menu items in preview mode
mFile.Enabled = False
mEdit.Enabled = False

Exit Sub

errHndl:
MsgBox "Error Previewing the Report: " & Err.Number & " " & Err.Description
End Sub

```

#### 4. Save and run the project.

Note: The samples contained in this section are designed to demonstrate the fundamentals for using the end-user report designer. More advanced samples can be found in the sample directory and in Data Dynamics• online knowledgebase at <http://www.datadynamics.com/kb>.

## Custom Toolbars and Menus

---

### Custom Toolbars and Menus

The runtime designer toolbars and menus cannot be customized during development. You can control the visibility and accessibility of individual toolbars using `ToolbarsVisible` and `ToolbarsAccessible` properties. You cannot remove any of the tools from the toolbars.

If you need to present your end users with a different user interface elements you should disable and hide all the toolbars by setting `ToolbarsVisible` and `ToolbarsAccessible` to 0 and create your own toolbars and menus.

`StatusChange` event and `ExecuteAction` and `QueryStatus` methods provide complete control over the current state of available UI options. In addition, you can customize the alerts and error messages by handling the `Alert` event.

In addition, you can create your own custom or localized object context menus in the `ContextMenuOpen` event.

The "Diamond Reports" sample included in your samples directory provides a comprehensive example for creating custom toolbars and menus.

### Included Sample Projects

---

#### Included Sample Projects

The ActiveReports Pro installation includes a few specialized sample projects to demonstrate the different techniques and capabilities available with the professional edition of ActiveReports.

The code behind the sample projects demonstrates many techniques available with the professional edition. Use these samples along with the following tutorial to help you understand the use of the various ActiveReports Professional components.

Following is a listing of these sample projects and the features they demonstrate:

#	Name	Description
1	Diamond Reports	An advanced project demonstrating the full possibilities of the run-time designer. Includes custom toolbars and menus that implement the functionality of the built-in counterparts.
2	Property List	Demonstrates using the property list box.
3	Simple Designer	Demonstrates using the run-time designer, property list box and preview form.

## Deployment and Distribution

---

### Deployment and Distribution

You need to include the following files on all clients when distributing ActiveReports Pro.

File Name	Description
Arpro2.DLL	The Reporting Engine.
ARVIEW2.ocx	Only if you are using our ActiveX Viewer.
ARdespro2.dll	Only if you are using the end-user report designer.
AB2DLL.dll	If you are using the run-time designer (not supported on the web).
PDFExpt.DLL	PDF Export Filter (when using PDF exporting).
RTFExpt.DLL	RTF Export Filter (when using RTF exporting).
ExclExpt.DLL	Excel Export Filter (when using Excel exporting).
TextExpt.DLL	Text Export Filter (when using Text exporting).
HTMLExpt.DLL	HTML Export Filter (when using HTML exporting).
TiffExpt.dll	Tiff Export Filter (when using Tiff exporting).
WebCache.dll	Only if you are using the WebCache service.

### Web Server Distribution

To serve reports to clients in a web environment, your web server should have arview2.cab if the project uses the ActiveReports Viewer Control and arpro2.cab if the project uses the end user designer control. You should also copy and register any export DLLs as needed.

### WebCache Service and ISAPI DLL

---

#### WebCache Service and ISAPI DLL

Introduction

Installation

Deployment

Using the WebCache Service

#### Introduction

---

#### Introduction

The WebCache service and ISAPI DLL are used to manage report output on web servers running Microsoft® Internet Information Servers. The caching service is a COM component that runs as service on the web server and caches the report's output. The ISAPI DLL receives requests for cache items, retrieves the items from the caching service and delivers them to the client browsers.

#### Installation

---

#### Installation

The setup program will automatically install WebCache.dll and WebCacheService.exe to your machine. The service defaults will be set to use the system account and automatic startup.

#### Deployment

---

## Deployment

To deploy the WebCacheService, you can add the WebCacheService.exe file to your setup project as a service or manually register the service using:

```
WebCacheService.exe -RegServer • Service
```

To uninstall, stop the service using the Control Panel / Administrative Tools / Services tool and then use

```
WebCacheService.exe -UnregServer
```

Note: If you are using Wise InstallMaster, don't use the service installation feature, instead add the following commands to your install script:

```
Execute Program %OCXPATH%\WebCacheService.exe -RegisterServer -Service
```

```
Add Execute path: %OCXPATH%\WebCacheService.exe • UnregServer to INSTALL.LOG
```

To configure the number of threads that the WebCacheService creates on startup set the Start Parameters /Threads=NumberOfThreads on the general property page of the service (Control Panel / Services).

## Using the WebCache Service

---

### Using the WebCache Service

The WebCache service can be utilized using either of the following methods:

1. CacheContent method allows you to cache any type of content including report output (RDF files) and export filters byte array output. The CacheContent method specifies the content type and the ISAPI filter would serve the cached items with the content and header specified in this method.
2. Excel and PDF Export Filters expose an ExportWebCache method that allows a direct export into the WebCache service objects and returns the proper cache item ids to redirect the client browser.

## Developers Reference

---

### Developers Reference

ActiveReports Run-time Designer

WebCache Service Objects

Property List Objects

### ActiveReports Run-time Designer

---

#### ActiveReports Run-time Designer

ARDesigner Control

Selection Object

#### ARDesigner

---

#### ARDesigner

Name	Type	Description
GridSnap	Property	Determines whether the controls should be snapped to the grid points.
GridVisible	Property	Determines whether the drawing grid should be visible.
GridX	Property	Determines how coarse the designer grid should be.
GridY	Property	Determines how coarse the designer grid should be.
IsDirty	Property	Returns whether report has been modified since last layout was loaded or initialized.
Locked	Property	Specifies whether the controls are locked in place.
Report	Property	Returns a reference to the designer's report object.
RulerUnits	Property	Sets or returns ruler units (Inches, Centimeters).
SelectedObjects	Property	Returns collection of selected objects.
ToolbarsAccessible	Property	Bit flags for each toolbar to determine whether a toolbar is accessible by the end user.
ToolbarsVisible	Property	Bit flags for each toolbar to determine whether a toolbar is visible.
ToolboxItem	Property	Sets or returns PROGID of active toolbox item.
ExecuteAction	Method	Executes a specified designer command. object.ExecuteAction(action As DesignerActionTypes)
LoadFromObject	Method	Reads the layout from a report object into designer control. object.LoadFromObject(Report As IActiveReport)
NewLayout	Method	Discards the current report layout and creates a new blank layout. object.NewLayout()
QueryStatus	Method	Queries the designer for the status of one or more commands. object.QueryStatus(action As DesignerActionTypes)
SaveToObject	Method	Write the layout from the designer to a report object. object.SaveToObject(Report As IActiveReport)
Alert	Event	Fires when an alert requesting user intervention is about to be displayed.
ContextMenuOpen	Event	Fires before a context menu is opened.
Error	Event	Fires when an error occurs in the designer component.
LayoutChanged	Event	Fires when the report layout is changed.
SelChange	Event	Fires when selection changes.
StatusChange	Event	Fires for each change in the status of the designer actions.
ValidateChange	Event	Fires before an item is moved, sized or deleted.

## Properties

---

### ARDesigner Properties

GridSnap

GridVisible

GridX

GridY

IsDirty

Locked

Report

RulerUnits

SelectedObjects

ToolbarsAccessible

ToolbarsVisible

ToolboxItem

### GridSnap

---

#### GridSnap

Determines whether the controls should be snapped to the grid points.

#### Syntax

object.GridSnap [= value]

The GridSnap property syntax has the following parts

Part	Description
object	A valid ARDesigner object
value	A Boolean value.

#### Settings

The settings for value are:

Setting	Description
True	Default - The controls are snapped to the grid points.
False	The controls can be sized and positioned freely.

#### Data Type

Boolean

#### Remarks

Default value = True

### GridVisible

---

#### GridVisible

Determines whether the drawing grid should be visible.

#### Syntax

object.GridVisible [= value]

The GridVisible property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Shows the grid in the designer.
False	Hides the grid in the designer.

### Data Type

Boolean

### Remarks

Default value = True

## GridX

---

### GridX

Determines how coarse the designer grid should be.

### Syntax

object.GridX [= value]

The GridX property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	An Integer value that represents the number of horizontal grid points per ruler unit.

### Data Type

Integer

### Remarks

Default value = 16

## GridY

---

### GridY

Determines how coarse the designer grid should be.

### Syntax

object.GridY [= value]

The GridY property syntax has the following parts

Part	Description
object	A valid ARDesigner object
value	An Integer value that represents the number of vertical grid points per ruler unit.

### Data Type

Integer

Remarks

Default value = 16

## IsDirty

---

### IsDirty

Returns whether report has been modified since last layout was loaded or initialized.

### Syntax

object.IsDirty [= value]

The IsDirty property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	The report layout has been modified.
False	The report layout has not been modified.

### Data Type

Boolean

### Example

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
    If ARDesigner1.IsDirty Then
        ' Ask if report should be saved
        Dim iSave As Integer
        iSave = MsgBox("Save changes to the report?", _
            vbYesNoCancel, "Save")
        Select Case iSave
            Case vbYes
                'Save the Report
                FileSave
                Cancel = 0
            Case vbNo
                'Continue without saving
                Cancel = 0
            Case vbCancel
                'Cancel Unload
                Cancel = 1
        End Select
    End If
End Sub
```

## Locked

---

### Locked

Specifies whether the controls are locked in place.

### Syntax

object.Locked [= value]

The Locked property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	The controls cannot be moved or sized.
False	The controls can be moved or sized.

### Data Type

Boolean

### Example

```
' If the controls are locked mark the menu item as checked
mnuLocked.Checked = ARDesigner1.Locked
```

### Remarks

Default value = False

## Report

---

### Report

Returns a reference to the designer's report object.

### Syntax

object.Report [= value]

The Report property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	An ActiveReport reference.

### Data Type

IActiveReport

### Example

```
'Add a data control to the designer using the Report object
Dim ctl As DataControl
With ARDesigner.Report.Sections("Detail").Controls
Set ctl = .Add("DDActiveReports2.DataControl")
    ctl.Name = "dc"
    ctl.Top = 0: ctl.Left = 0
    ctl.Tag = ""
End With
```

### Remarks

This report object is used to gain access to the layout and controls properties. Do not use it to run the report and preview it. Use a separate ActiveReport variable and save the layout to it using the SaveToObject method.



## RulerUnits

---

### RulerUnits

Sets or returns ruler units (Inches, Centimeters).

### Syntax

object.RulerUnits [= value]

The RulerUnits property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	A valid ddRulerUnits setting.

### Settings

The settings for value are:

Setting	Description
ddRulerUS	0 - Inches.
ddRulerMetric	1 - Centimeters.

### Data Type

ddRulerUnits

### Remarks

Default value = 0 - US Setting.

### SelectedObjects

---

### SelectedObjects

Returns collection of selected objects.

### Syntax

Set value = object.SelectedObjects

The SelectedObjects property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	A Selection object.

### Data Type

Selection

### Example

```
Private Sub ARDesigner1_SelChange()
    Dim lSel As Long
    Dim arrSel()
    'plist is a custom PropertyList control
    plist.Clear

    'When selection changes, add selected objects to the custom
    'property list
    If ARDesigner1.SelectedObjects.Count > 0 Then
        ReDim arrSel(ARDesigner1.SelectedObjects.Count - 1)
```

```

        For lSel = 0 To ARDesigner1.SelectedObjects.Count - 1
            Set arrSel(lSel) = ARDesigner1.SelectedObjects(lSel)
        Next
        plist.SelectObjects arrSel
    End If
End Sub

```

## ToolbarsAccessible

---

### ToolbarsAccessible

Bit flags for each toolbar to determine whether a toolbars is accessible by the end user. One additional flag for the context menus or a property to enable or disable the context menus.

#### Syntax

object.ToolbarsAccessible [= value]

The ToolbarsAccessible property syntax has the following parts

Part	Description
object	A valid ARDesigner object
value	A ToolbarIdentifiers setting.

#### Settings

The settings for value are:

Setting	Description
ddTBMenu	1 - Main menu toolbar.
ddTBToolBox	2 - Controls toolbox.
ddTBStandard	4 - Standard toolbar.
ddTBAlignment	8 - Alignment toolbar.
ddTBFormat	16 - Format toolbar.
ddTBExplorer	32 - Report explorer toolbar.
ddTBFields	64 - Fields list toolbar.
ddTBPropertyToolbox	128 - Property toolbox.

#### Data Type

ToolbarIdentifiers

#### Example

```

Private Sub Form_Load()
    'Disable and hide the built-in toolbars
    ARDesigner1.ToolbarsAccessible = 0
    ARDesigner1.ToolbarsVisible = 0
End Sub

```

#### Remarks

The customization option for the toolbars is available only when all toolbars are accessible. If any of the toolbars is not accessible the built-in customization will be disabled.

## ToolbarsVisible

---

### ToolbarsVisible

Bit flags for each toolbar to determine whether a toolbar is visible. The end user can show/hide the toolbars from the toolbar's context menu.

## Syntax

object.ToolbarsVisible [= value]

The ToolbarsVisible property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	A ToolbarIdentifiers setting.

## Settings

The settings for value are:

Setting	Description
ddTBMenu	1 - Main menu toolbar.
ddTBToolBox	2 - Controls toolbox.
ddTBStandard	4 - Standard toolbar.
ddTBAlignment	8 - Alignment toolbar.
ddTBFormat	16 - Format toolbar.
ddTBExplorer	32 - Report explorer toolbar.
ddTBFields	64 - Fields list toolbar.
ddTBPropertyToolbox	128 - Property toolbox.

## Data Type

ToolbarIdentifiers

## Example

```
Private Sub Form_Load()
    'Disable and hide the built-in toolbars
    ARDesigner1.ToolbarsAccessible = 0
    ARDesigner1.ToolbarsVisible = 0
End Sub
```

## ToolboxItem

### ToolboxItem

Sets or returns PROGID of active toolbox item. Set to empty to end control mode.

### Syntax

object.ToolBoxItem [= value]

The ToolBoxItem property syntax has the following parts:

Part	Description
object	A valid ARDesigner object
value	A String value.

### Data Type

String

### Example

```
Private Sub tbToolbox_ButtonClick(ByVal Button As MSComctlLib.Button)
    Select Case Button.key
        Case "tbxSelect": ARDesigner1.ToolBoxItem = ""
        Case "tbxLabel": ARDesigner1.ToolBoxItem = "DDActiveReports2.Label"
        Case "tbxField": ARDesigner1.ToolBoxItem = "DDActiveReports2.Field"
    End Select
End Sub
```

```

Case "tbxCheckbox": ARDesigner1.ToolBoxItem = "DDActiveReports2.Checkbox"
Case "tbxImage": ARDesigner1.ToolBoxItem = "DDActiveReports2.Image"
Case "tbxLine": ARDesigner1.ToolBoxItem = "DDActiveReports2.Line"
Case "tbxShape": ARDesigner1.ToolBoxItem = "DDActiveReports2.Shape"
Case "tbxRichedit": ARDesigner1.ToolBoxItem = "DDActiveReports2.RichEdit"
Case "tbxFrame": ARDesigner1.ToolBoxItem = "DDActiveReports2.Frame"
Case "tbxSubreport": ARDesigner1.ToolBoxItem = "DDActiveReports2.Subreport"
Case "tbxPageBreak": ARDesigner1.ToolBoxItem = "DDActiveReports2.PageBreak"
Case "tbxOLE": ARDesigner1.ToolBoxItem = "DDActiveReports2.OLE"
Case "tbxBarcode": ARDesigner1.ToolBoxItem = "DDActiveReports2.Barcode"
End Select
End Sub

```

## Remarks

This property is used to implement a custom toolbox toolbar.

## Methods

---

### ARDesigner Properties

ExecuteAction  
 GetSectionFromPoint  
 LoadFromObject  
 NewLayout  
 QueryStatus  
 SaveToObject

### ExecuteAction

---

#### ExecuteAction

Executes a specified designer command. You can use this method when implementing a custom toolbar or menu, this method will perform the report actions in response to the toolbar or menu items.

#### Syntax

```
object.ExecuteAction(action As DesignerActionTypes)
```

The ExecuteAction method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type ARDesigner.
action	DesignerActionTypes - A valid action setting.

#### Settings

The settings for action are:

Setting	Description
ddActionFOpen	1 - File: Open.
ddActionFSave	2 - File: Save.
ddActionFPageSetup	3 - File: Page Setup.
ddActionECut	4 - Edit: Cut.
ddActionEPaste	5 - Edit: Paste.
ddActionECopy	6 - Edit: Copy.
ddActionEUndo	7 - Edit: Undo.
ddActionEDelete	8 - Edit: Delete.
ddActionEDeleteSection	9 - Edit: Delete Section.
ddActionEInsertReportHF	10 - Edit: Insert Report Header/Footer.

ddActionEInsertPageHF	11 - Edit: Insert Page Header/Footer.
ddActionEInsertGroupHF	12 - Edit: Insert Group Header/Footer.
ddActionEReorderGroups	13 - Edit: Reorder Groups.
ddActionEInsertField	14 - Edit: Insert Field.
ddActionViewExplorer	15 - View: Report Explorer.
ddActionViewFieldsList	16 - View: Fields List.
ddActionViewPropertyList	17 - View: Property Listbox.
ddActionViewGrid	18 - View: Grid.
ddActionViewSnapToGrid	19 - View: Snap to grid.
ddActionViewFullScreen	20 - View: Full screen.
ddActionViewCodeEditor	21 - View: Script Code Editor.
ddActionFoAlignLefts	22 - Format: Align Control Lefts.
ddActionFoAlignRights	23 - Format: Align Control Rights.
ddActionFoAlignCenters	24 - Format: Align Control Centers.
ddActionFoAlignTops	25 - Format: Align Control Tops.
ddActionFoAlignMiddles	26 - Format: Align Control Middles.
ddActionFoAlignBottoms	27 - Format: Align Control Bottoms.
ddActionFoAlignToGrid	28 - Format: Align to Controls Grid.
ddActionFoAlignCenterInSec	29 - Format: Align : Center Control in Section.
ddActionFoSizeSameWidth	30 - Format: Size controls to the same width.
ddActionFoSizeSameHeight	31 - Format: Size controls to the same height.
ddActionFoSizeSameBoth	32 - Format: Size controls to the same width and height.
ddActionFoVSpaceEqual	33 - Format: Space controls even vertically.
ddActionFoVSpaceIncrease	34 - Format: Increase vertical spacing.
ddActionFoVSpaceDecrease	35 - Format: Decrease vertical spacing.
ddActionFoHSpaceEqual	36 - Format: Space controls even horizontally.
ddActionFoHSpaceIncrease	37 - Format: Increase horizontal spacing.
ddActionFoHSpaceDecrease	38 - Format: Decrease horizontal spacing.
ddActionFoOrderBringToFront	39 - Format: Bring control to the foreground.
ddActionFoOrderSendToBack	40 - Format: Send control to the background.
ddActionFoLockControls	41 - Format: Lock controls size and position.
ddActionFoFontBold	45 - Format: bold.
ddActionFoFontItalic	46 - Format: Italic.
ddActionFoTextAlignLeft	47 - Format: Align text left.
ddActionFoTextAlignCenter	48 - Format: Align text center.
ddActionFoTextAlignRight	49 - Format: Align text Right.
ddActionFoBorder	54 - Format: Set border styles.
ddActionFoBullets	55 - Format: Set bullet style.
ddActionFoIndent	56 - Format: Indent text.
ddActionFoOutdent	57 - Format: Outdent text.
ddActionFoUnderline	58 - Format: Underline.

### Example

```
'Edit > Cut menu item
Private Sub miECut_Click()
    ARDesigner1.ExecuteAction ddActionECut
End Sub
```

### Remarks

Font and color actions are not supported in the ExecuteAction method. In order to set font and color properties you should directly access the selected object and set those properties.

### GetSectionFromPoint

### GetSectionFromPoint

Returns the section name at a specified point and converts the point coordinates to section relative coordinates. Returns empty when the specified point is not within a section area.

### Syntax

```
[sectionName = ]object.GetSectionFromPoint(x As Single, y As Single)
```

The GetSectionFromPoint method syntax has the following parts:

Part	Description
object	A valid ARDesigner object.
x, y	Single - Specifies the point coordinates of which to retrieve the section name. The values are converted to section relative coordinates on return from the method.
sectionName String	Returns the section name that is at the specified point coordinates.

### Returns

String

### Example

```
Private deltax As Single, deltay As Single
' This code implements a label Drag Drop on the designer control.
' It adds a new control at the dropped location.
Private Sub ard_DragDrop(Source As Control, X As Single, Y As Single)
    Dim sSec As String
    Dim secTarget As Object
    Dim ctl As Object

    X = X - deltax
    Y = Y - deltay

    sSec = ard.GetSectionFromPoint(X, Y)
    If sSec <> "" Then
        Set secTarget = ard.Report.Sections(sSec)
        Set ctl = secTarget.Controls.Add("DDActiveReports2.Label")
        ctl.Left = X
        ctl.Top = Y
        ctl.Width = lblDrag.Width
        ctl.Height = lblDrag.Height
        ctl.BackStyle = 1
        ctl.BackColor = &HC0C0FF
        If (ctl.Left + ctl.Width) > ard.Report.PrintWidth Then
            ard.Report.PrintWidth = ctl.Left + ctl.Width
        End If
        If (ctl.Top + ctl.Height) > secTarget.Height Then
            secTarget.Height = ctl.Top + ctl.Height
        End If
    End If
End Sub

Private Sub ard_DragOver(Source As Control, X As Single, Y As Single, State As Integer)
    Dim sSec As String
    X = X - deltax
    Y = Y - deltay

    sSec = ard.GetSectionFromPoint(X, Y)
    lstState.AddItem sSec & " : " & Str$(X) & "," & Str$(Y)
End Sub
```

### Remarks

This method is used when adding controls into specific sections using drag and drop events.

### LoadFromObject

---

## LoadFromObject

Reads the layout from a report object into designer control.

### Syntax

```
object.LoadFromObject(Report As IActiveReport)
```

The LoadFromObject method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type ARDesigner.
Report	IActiveReport

### Example

```
'Load a report layout file into an activereport instance
'then load it into the designer control.
'Alternatively, you can use the Load method of the deisgner's Report property
Dim rpt As ActiveReport
Set rpt = New ActiveReport
rpt.Load App.Path & "\test.rpx"
ARDesigner1.LoadFromObject(rpt)
```

## NewLayout

---

### NewLayout

Discards the current report layout and creates a new blank layout.

### Syntax

```
object.NewLayout()
```

The NewLayout method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type ARDesigner.

### Example

```
'File > New, Menu Item
Private Sub miFNew_Click()
    ARDesigner1.NewLayout
End Sub
```

## QueryStatus

---

### QueryStatus

Queries the object for the status of one or more commands.

### Syntax

```
object.QueryStatus(action As DesignerActionTypes)
```

The QueryStatus method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type ARDesigner.
action	DesignerActionTypes

### Settings

The settings for action are:

Setting	Description
ddActionFOpen	1 - File: Open.
ddActionFSave	2 - File: Save.
ddActionFPageSetup	3 - File: Page Setup.
ddActionECut	4 - Edit: Cut.
ddActionEPaste	5 - Edit: Paste.
ddActionECopy	6 - Edit: Copy.
ddActionEUndo	7 - Edit: Undo.
ddActionEDelete	8 - Edit: Delete.
ddActionEDeleteSection	9 - Edit: Delete Section.
ddActionEInsertReportHF	10 - Edit: Insert Report Header/Footer.
ddActionEInsertPageHF	11 - Edit: Insert Page Header/Footer.
ddActionEInsertGroupHF	12 - Edit: Insert Group Header/Footer.
ddActionEReorderGroups	13 - Edit: Reorder Groups.
ddActionEInsertField	14 - Edit: Insert Field.
ddActionViewExplorer	15 - View: Report Explorer.
ddActionViewFieldsList	16 - View: Fields List.
ddActionViewPropertyList	17 - View: Property Listbox.
ddActionViewGrid	18 - View: Grid.
ddActionViewSnapToGrid	19 - View: Snap to grid.
ddActionViewFullScreen	20 - View: Full screen.
ddActionViewCodeEditor	21 - View: Script Code Editor.
ddActionFoAlignLefts	22 - Format: Align Control Lefts.
ddActionFoAlignRights	23 - Format: Align Control Rights.
ddActionFoAlignCenters	24 - Format: Align Control Centers.
ddActionFoAlignTops	25 - Format: Align Control Tops.
ddActionFoAlignMiddles	26 - Format: Align Control Middles.
ddActionFoAlignBottoms	27 - Format: Align Control Bottoms.
ddActionFoAlignToGrid	28 - Format: Align to Controls Grid.
ddActionFoAlignCenterInSec	29 - Format: Align : Center Control in Section.
ddActionFoSizeSameWidth	30 - Format: Size controls to the same width.
ddActionFoSizeSameHeight	31 - Format: Size controls to the same height.
ddActionFoSizeSameBoth	32 - Format: Size controls to the same width and height.
ddActionFoVSpaceEqual	33 - Format: Space controls even vertically.
ddActionFoVSpaceIncrease	34 - Format: Increase vertical spacing.
ddActionFoVSpaceDecrease	35 - Format: Decrease vertical spacing.
ddActionFoHSpaceEqual	36 - Format: Space controls even horizontally.
ddActionFoHSpaceIncrease	37 - Format: Increase horizontal spacing.
ddActionFoHSpaceDecrease	38 - Format: Decrease horizontal spacing.
ddActionFoOrderBringToFront	39 - Format: Bring control to the foreground.
ddActionFoOrderSendToBack	40 - Format: Send control to the background.
ddActionFoLockControls	41 - Format: Lock controls size and position.
ddActionFoStyle	42 - Format: Style.
ddActionFoFontName	43 - Format: Font name.
ddActionFoFontSize	44 - Format: Font size.
ddActionFoFontBold	45 - Format: bold.
ddActionFoFontItalic	46 - Format: Italic.
ddActionFoTextAlignLeft	47 - Format: Align text left.
ddActionFoTextAlignCenter	48 - Format: Align text center.
ddActionFoTextAlignRight	49 - Format: Align text Right.
ddActionFoForeColor	50 - Format: Set foreground color.
ddActionFoBackColor	51 - Format: Set background color.
ddActionFoLineStyle	52 - Format: Set line style.
ddActionFoLineColor	53 - Format: Set line color.



ddActionFoBorder	54 - Format: Set border styles.
ddActionFoBullets	55 - Format: Set bullet style.
ddActionFoIndent	56 - Format: Indent text.
ddActionFoOutdent	57 - Format: Outdent text.
ddActionFoUnderline	58 - Format: Underline.

### Example

```
'Update edit menu items on status change.
Private Sub ARDesigner1_StatusChange(ByVal action As DDActiveReportsDesignerCtl.DesignerActic
    Select Case action
        Case ddActionECopy
            miECopy.Enabled = ((ARDesigner1.QueryStatus(ddActionECopy) And d
            miECopy.Checked = ((ARDesigner1.QueryStatus(ddActionECopy) And d
        ' Case ....
    End Select
End Sub
```

## SaveToObject

---

### SaveToObject

Write the layout from the designer to a report object.

#### Syntax

```
object.SaveToObject(Report As IActiveReport)
```

The SaveToObject method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type ARDesigner.
Report	IActiveReport

### Example

```
'module variable
Dim rpt As DDActiveReports2.ActiveReport

Private Sub PreviewReport()
    On Error GoTo ehPreviewReport
    ard.SaveToObject rpt
    rpt.Restart
    rpt.Run False
    Set arv.ReportSource = rpt
    Exit Sub

    ehPreviewReport:
    MsgBox Str(Err.Number) & " - " & Err.Description, vbOKOnly, "Error: PreviewReport"
End Sub
```

### Remarks

You must use the SaveToObject to save the report designer to an ActiveReport instance before running the report.

### Events

---

## ARDesigner Properties

Alert

ContextMenuOpen

Error  
 LayoutChanged  
 SelChange  
 StatusChange  
 ValidateChange

## Alert

---

### Alert

Fires when before an alert message box that requires user intervention is displayed. You can use this event to replace the built-in message boxes with your own.

### Syntax

Sub object\_Alert(id As Integer, prompt As String, buttons As Long, result)

The Alert event syntax has the following parts:

Part	Description
id	Integer - Specifies the alert message id.
prompt	String - Specifies the message string to be displayed.
buttons	Long - Specifies the number and style of buttons to be displayed.
result	Long - used to set the return value of the event when the alert is handled by the event.

### Settings

The id parameter has the following settings:

Setting	Description
ddARAlertControlNotRegistered	1 - Report contains a control that is not registered on the client machine.
ddARAlertDataSource	2 - Data source returned error when updating property sheet.
ddARAlertDAOSettings	3 - DAO data control settings are incorrect.
ddARAlertDAO	4 - DAO returned error when opening the connection or recordset.
ddARAlertFieldList	5 - A database error occurred when attempting to refresh the field list window.
ddARAlertInvalidSectionForDataControl	6 - A data control cannot be added to a non-detail section.
ddARAlertDataControlAlreadyExists	7 - User tried to drop more than one data control into the detail section.
ddARAlertControlCreateFailed	8 - The ActiveX control can't be hosted in ActiveReports
ddARAlertAB2DLLMissing	9 - AB2DLL.DLL toolbars library is missing.
ddARAlertCantUndoDelete	10 - The edit/delete operation can't be undone
ddARAlertDeleteFailed	11 - The edit/delete operation failed.
ddARAlertEditCutFailed	12 - The edit/cut operation failed.
ddARAlertEditCopyFailed	13 - The edit/copy operation failed.
ddARAlertDuplicateStyleName	14 - User tried to create a style that already exists.
ddAlertCantDeleteStyle	15 - User tried to delete the normal style.
ddAlertRTF	16 - RTF control alert.
ddARAlertRTFDeleteField	17 - Confirm deleting an RTF merge field.
ddARAlertCantDeleteDetailSection	18 - Detail section cannot be deleted.
ddARAlertDeleteSectionPrompt	19 - Confirm deleting a section.
ddARAlertSaveLayoutFailed	20 - Unable to save the report layout.

## Example

```
Private Sub ard_Alert(ByVal id As Integer, ByVal prompt As String, ByVal buttons As Long, res
    If id = ddARAlertControlNotRegistered Then
        MsgBox "Report contains an unregistered control." & _
            "Contact 999-999-9999 with the following information " & _
            vbCrLf & Str(id) & " - " & prompt
        result = 0
    End If
End Sub
```

## ContextMenuOpen

---

### ContextMenuOpen

Fires before a context menu is opened.

#### Syntax

```
object_ContextMenuOpen(sourceObject As Object,
    menuType As ContextMenuTypes,
    Cancel As Boolean)
```

The ContextMenuOpen event syntax has the following parts:

Part	Description
sourceObject Object	A reference to the object that is opening the menu.
menuType ContextMenuTypes	Specifies the type of menu that will be opened for this sourceObject.
Cancel Boolean	determines whether the default menu handler should be cancelled. This parameter should be set to True to disable or replace built in context menus.

#### Settings

The settings for menuType are:

Setting	Description
ddCMSection	0 - Section context menu.
ddCMControl	1 - Control context menu.
ddCMReport	2 - Report object context menu.
ddCMRTFEditMode	3 - RichEdit context menu.

## Example

```
'Example implementation of the ContextMenuOpen event
'The mnuReport, mnuControl, mnuSection and mnuRichEdit
'are menu items created using VB's Menu editor
'You can use the sourceObject properties to enable/disable
'your custom menu options
Private Sub ARDesigner1_ContextMenuOpen(ByVal sourceObject As Object, ByVal menuType As DDAct
    Select Case menuType
        Case ddCMControl
            PopupMenu mnuControl
        Case ddCMReport
            PopupMenu mnuReport
        Case ddCMSection
            PopupMenu mnuSection
        Case ddCMRichedit
            PopupMenu mnuRichEdit
    End Select
    Cancel = True
```

End Sub

## Error

---

### Error

Fires when an error occurs in the designer component. This event allows you to create your own error handler and display localized error message boxes.

### Syntax

```
object_Error((Number As Integer, Description As String, Scode As Long, Source As String, HelpFile As String, HelpContext As Long, CancelDisplay As Boolean))
```

The Error event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type ARDesigner.
Number	Integer - Error number
Description	String - Error description.
Scode	Long - Result code.
Source	String - Source of the error if applicable.
HelpFile	String - Help file
HelpContext	Long - Error context id, in the help file.
CancelDisplay	Boolean - Set CancelDisplay = True to cancel the built in error dialog and replace it with your own.

### Example

```
Private Sub ARDesigner1_Error(ByVal Number As Integer, Description As String, _
    ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, _
    ByVal HelpContext As Long, CancelDisplay As Boolean)
    App.LogEvent Format(Now, "mm/dd/yyyy Hh:Nn") & Str(Number) & " - " & Description
    CancelDisplay = True
End Sub
```

## LayoutChanged

---

### LayoutChanged

Fires when the layout is changed. You can use this event to monitor changes to the report layout and update any dependent data such as SQL queries or custom user interfaces (report explorers, group sections dialog, etc.)

### Syntax

```
object_LayoutChanged(changedObject As Object, changeType As LayoutChangeTypes)
```

The LayoutChanged event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type ARDesigner.
changedObject	Object - a reference to the control or object that caused the layout change.
changeType	LayoutChangeTypes - specifies the type of change.

### Settings

The settings for changeType are:

Setting	Description
ddLCControlMove	0 - A control's position has changed.
ddLCControlSize	1 - A control's size has changed.
ddLCControlDelete	2 - A control is deleted.
ddLCSectionSize	3 - A section's size has changed.
ddLCSectionDelete	4 - A section is deleted.
ddLCSectionMove	5 - A section is moved.
ddLCReportSize	6 - The report's size is changed.
ddLCControlAdd	7 - A new control is added to the report.
ddLCSectionsAdd	8 - A section is added.

### Example

```
Private Sub ARDesigner1_LayoutChanged(changedObject As Object, _
    changeType As LayoutChangeTypes)
    'If a group section was added or removed then display a grouping dialog
    If changeType = ddLCSectionAdd Then
        If changedObject.Type = ddSTGroupHeader Then
            frmGroups.Show
        End If
    End If
End Sub
```

## SelChange

---

### SelChange

Fires when selection changes. You can use the SelectedObjects property to inspect the current selection.

#### Syntax

object\_SelChange()

#### Example

```
'SelChange event handler
Private Sub ARDesigner1_SelChange()
    If ARDesigner1.SelectedObjects.Count = 1 Then
        StatusBar1.Panels(1).Text = ARDesigner1.SelectedObjects(0).Name
    Else
        StatusBar1.Panels(1).Text = ""
    End If
End Sub
```

#### Remarks

This event can be used to update UI elements such as a property toolbox or status bar in your custom report designer.

## StatusChange

---

### StatusChange

This event fires for each change in the status of the designer actions. Designer actions represent the commands that are typically invoked from UI elements such as a toolbar or a menu. You can use the QueryStatus method to check the status of the changed action and update your custom UI elements.

#### Syntax

object\_StatusChange(action As DesignerActionTypes)

The StatusChange event syntax has the following parts:

Part	Description
actionDesignerActionTypes	- Specifies the action that caused the change as one of the actions listed below.

## Settings

The settings for action are:

Setting	Description
ddActionFOpen	1 - File: Open.
ddActionFSave	2 - File: Save.
ddActionFPageSetup	3 - File: Page Setup.
ddActionECut	4 - Edit: Cut.
ddActionEPaste	5 - Edit: Paste.
ddActionECopy	6 - Edit: Copy.
ddActionEUndo	7 - Edit: Undo.
ddActionEDelete	8 - Edit: Delete.
ddActionEDeleteSection	9 - Edit: Delete Section.
ddActionEInsertReportHF	10 - Edit: Insert Report Header/Footer.
ddActionEInsertPageHF	11 - Edit: Insert Page Header/Footer.
ddActionEInsertGroupHF	12 - Edit: Insert Group Header/Footer.
ddActionEReorderGroups	13 - Edit: Reorder Groups.
ddActionEInsertField	14 - Edit: Insert Field.
ddActionViewExplorer	15 - View: Report Explorer.
ddActionViewFieldsList	16 - View: Fields List.
ddActionViewPropertyList	17 - View: Property Listbox.
ddActionViewGrid	18 - View: Grid.
ddActionViewSnapToGrid	19 - View: Snap to grid.
ddActionViewFullScreen	20 - View: Full screen.
ddActionViewCodeEditor	21 - View: Script Code Editor.
ddActionFoAlignLefts	22 - Format: Align Control Lefts.
ddActionFoAlignRights	23 - Format: Align Control Rights.
ddActionFoAlignCenters	24 - Format: Align Control Centers.
ddActionFoAlignTops	25 - Format: Align Control Tops.
ddActionFoAlignMiddles	26 - Format: Align Control Middles.
ddActionFoAlignBottoms	27 - Format: Align Control Bottoms.
ddActionFoAlignToGrid	28 - Format: Align to Controls Grid.
ddActionFoAlignCenterInSec	29 - Format: Align : Center Control in Section.
ddActionFoSizeSameWidth	30 - Format: Size controls to the same width.
ddActionFoSizeSameHeight	31 - Format: Size controls to the same height.
ddActionFoSizeSameBoth	32 - Format: Size controls to the same width and height.
ddActionFoVSpaceEqual	33 - Format: Space controls even vertically.
ddActionFoVSpaceIncrease	34 - Format: Increase vertical spacing.
ddActionFoVSpaceDecrease	35 - Format: Decrease vertical spacing.
ddActionFoHSpaceEqual	36 - Format: Space controls even horizontally.
ddActionFoHSpaceIncrease	37 - Format: Increase horizontal spacing.
ddActionFoHSpaceDecrease	38 - Format: Decrease horizontal spacing.
ddActionFoOrderBringToFront	39 - Format: Bring control to the foreground.
ddActionFoOrderSendToBack	40 - Format: Send control to the background.
ddActionFoLockControls	41 - Format: Lock controls size and position.
ddActionFoStyle	42 - Format: Style.
ddActionFoFontName	43 - Format: Font name.
ddActionFoFontSize	44 - Format: Font size.
ddActionFoFontBold	45 - Format: bold.
ddActionFoFontItalic	46 - Format: Italic.
ddActionFoTextAlignLeft	47 - Format: Align text left.

ddActionFoTextAlignCenter	48 - Format: Align text center.
ddActionFoTextAlignRight	49 - Format: Align text Right.
ddActionFoForeColor	50 - Format: Set foreground color.
ddActionFoBackColor	51 - Format: Set background color.
ddActionFoLineStyle	52 - Format: Set line style.
ddActionFoLineColor	53 - Format: Set line color.
ddActionFoBorder	54 - Format: Set border styles.
ddActionFoBullets	55 - Format: Set bullet style.
ddActionFoIndent	56 - Format: Indent text.
ddActionFoOutdent	57 - Format: Outdent text.
ddActionFoUnderline	58 - Format: Underline

## Example

```
'Update edit menu items on status change.
Private Sub ARDesigner1_StatusChange(ByVal action As DDActiveReportsDesignerCtl.DesignerActi
    Select Case action
        Case ddActionECopy
            miECopy.Enabled = ((ARDesigner1.QueryStatus(ddActionECopy) And _
                ddStatEnabled) = ddStatEnabled)
            miECopy.Checked = ((ARDesigner1.QueryStatus(ddActionECopy) And _
                ddStatChecked) = ddStatChecked)
    End Select
End Sub
```

## ValidateChange

---

### ValidateChange

This event is fired before an item is moved, sized or deleted. You can use this event to control the end user's actions. For example, you can prevent the user from deleting the report's data control or moving a predefined set of controls that are part of a standard report template.

### Syntax

```
object.ValidateChange(control As Object, changeType As LayoutChangeTypes, Cancel As Boolean)
```

### Parameters

The ValidateChange event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type ARDesigner.
control	Object
changeType	LayoutChangeTypes
Cancel	Boolean

### Settings

The settings for changeType are:

Setting	Description
ddLCControlMove	0 - A control's position has changed.
ddLCControlSize	1 - A control's size has changed.
ddLCControlDelete	2 - A control is deleted.
ddLCSectionSize	3 - A section's size has changed.
ddLCSectionDelete	4 - A section is deleted.
ddLCSectionMove	5 - A section is moved.
ddLCReportSize	6 - The report's size is changed.
ddLCControlAdd	7 - A new control is added to the report.

## Example

```
Private Sub ARDesigner1_ValidateChange(ByVal control As Object, _
    ByVal changeType As DDActiveReportsDesignerCtl.LayoutChangeTypes, _
    Cancel As Boolean)
    If changeType = ddLCControlDelete Then
        If control.Name = "DataControl1" Then
            MsgBox "You cannot delete the reports data source."
            Cancel = True
        End If
    End If
End Sub
```

## Selection Methods

---

### Selection Methods

Name	Type	Description
Count	Method	Returns the number of selection objects in the collection. object.Count
Item	Method	Returns the object at the selected index. object.Item(index)

## Count

---

### Count

Returns the number of selected objects in the collection.

#### Syntax

```
object.Count()
```

#### Example

```
'SelChange event handler
Private Sub ARDesigner1_SelChange()
    If ARDesigner1.SelectedObjects.Count = 1 Then
        StatusBar1.Panels(1).Text = ARDesigner1.SelectedObjects(0).Name
    Else
        StatusBar1.Panels(1).Text = ""
    End If
End Sub
```

## Item

---

### Item

Returns the selection item at the specified index.

#### Syntax

```
object.Item((index As Long))
```

The Item method syntax has the following parts:

Part	Description
------	-------------



object An expression evaluating to an object of type Selection.  
 index Long

### Example

```
'SelChange event handler
Private Sub ARDesigner1_SelChange()
    If ARDesigner1.SelectedObjects.Count = 1 Then
        StatusBar1.Panels(1).Text = ARDesigner1.SelectedObjects.Item(0).Name
    Else
        StatusBar1.Panels(1).Text = ""
    End If
End Sub
```

## WebCache Service

---

### WebCache Service Objects

WebCache

WebCacheItem

WebCacheWorkerThread

WebCacheWorkerThreads

### WebCache

---

#### WebCache

Name	Type	Description
CacheContent	Method	Adds an item to the WebCache collection.
CacheItem	Method	Adds an item to the WebCache collection.
IsCached	Method	Determines whether a specific item is cached.
Item	Method	Returns the cached item at the specified index.
Remove	Method	Removes the cached item at the specified index.
RemoveAll	Method	Removes all cached items from the service.
Count	Property	Returns the number of cached items in the service.

## CacheContent

---

### CacheContent

Adds an item to the WebCache collection.

Note: CacheContent is the most commonly used method to add items to the WebCache collection. The CacheItem Method should only be used when additional header information other than content type needs to be written into the header of the cached item.

### Syntax

```
object.CacheContent(ContentType As String, Data As Variant)
```

The CacheContent method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type WebCache.
ContentType	String
Data	Variant

### Example

```
'The following example performs the following
'1)Loads an ActiveReport from a presaved XML file
'2)Runs the report
'3)Exports the report to a byte array in PDF format
'4)Adds the byte array to ActiveReports WebCache so
'that it may be streamed directly to the browser
'The example code is placed in a user-defined function.
'A typical scenario would be for this function to be placed
'in a COM object and called from an ASP page.
'You could then do an ASP response.redirect to the
'url where the pdf export was cached.
Public Function ExportReport() as long

    Dim rpt As ActiveReport
    Dim aWebCache As WebCache
    Dim pdfExpt As ActiveReportsPDFExport.ARExportPDF
    Dim PDFByteArray As Variant

    Set rpt = New ActiveReport
    Set aWebCache = New WebCache
    Set pdfExpt = New ActiveReportsPDFExport.ARExportPDF

    rpt.Load "c:\testing.rpx"

    rpt.run

    Call pdfExpt.ExportStream(rpt.Pages, PDFByteArray)

    lWebCacheID = aWebCache.CacheContent("Application/PDF", PDFByteArray)

    ExportReport = lWebCacheID 'lWebCacheID can now be used to access the cached pdf fi
'ASP Code calling the above function
Dim vWebCacheID
'vWebCacheID = arptserver.ExportReport()
'Response.Redirect "mywebsite/webcache.dll?" & vWebCacheID & "?"
End Function
```

## CacheItem

---

### CacheItem

Adds an item to the WebCache collection.

Note: CacheContent is the most commonly used method to add items to the WebCache collection. The CacheItem Method should only be used when additional header information other than content type needs to be written into the header of the cached item.

### Syntax

```
object.CacheItem(Header As String, Data As Variant)
```

The CacheItem method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type WebCache.
Header	String - A valid header string to send to the browser client.
Data	Variant - cache content.

## Example

```
'The following example performs the following
'1)Loads an ActiveReport from a presaved XML file
'2)Runs the report
'3)Exports the report to a byte array in PDF format
'4)Adds the byte array to ActiveReports WebCache so
'that it may be streamed directly to the browser
'The example code is placed in a user-defined function.
'A typical scenario would be for this function to be placed
'in a COM object and called from an ASP page.
'You could then do an ASP response.redirect to the
'url where the pdf export was cached.
Public Function ExportReport() as long

    Dim rpt As ActiveReport
    Dim aWebCache As WebCache
    Dim pdfExpt As ActiveReportsPDFExport.ARExportPDF
    Dim PDFByteArray As Variant

    Set rpt = New ActiveReport
    Set aWebCache = New WebCache
    Set pdfExpt = New ActiveReportsPDFExport.ARExportPDF

    rpt.Load "c:\testing.rpx"

    rpt.run

    Call pdfExpt.ExportStream(rpt.Pages, PDFByteArray)

    lWebCacheID = aWebCache.CacheContent("Application/PDF", PDFByteArray)

    ExportReport = lWebCacheID 'lWebCacheID can now be used to access the cached pdf fi
    'ASP Code calling the above function
    Dim vWebCacheID
    'vWebCacheID = arptserver.ExportReport()
    'Response.Redirect "mywebsite/webcache.dll?" & vWebCacheID & "?"
End Function
```

## IsCached

---

### IsCached

Returns a Boolean value telling the developer if a specific item is still cached or not.

#### Syntax

```
object.IsCached(Id As String)
```

The IsCached method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type WebCache.
Id	String

#### Example

```
'Checking to see if a webcache id is still cached
Dim blnIsCached As Boolean
Dim aWebCache As WebCache

Set aWebCache = New WebCache
blnIsCached=aWebCache.IsCached("1")
```

## Item

---

### Item

Allows random access to individual nodes within the WebCache collection.

#### Syntax

```
object.Item((Index As Variant))
```

The Item method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type WebCache.
Index	Variant

#### Example

```
'The example code demonstrates how to loop
'through all of the items in the
'webcache collection and
'print out each items timeout value
'Please Note that For EACH is not used in the example.
'The _NewEnum property of the webcache collection
'is not supported at this time so you cannot use For Each.

Dim x As Integer
For x = 0 To aWebCache.Count - 1
    Debug.Print "awebcache.item(" & x & ").timeout = " & aWebCache.Item(x).Timeout
Next
```

## Remove

---

### Remove

Removes an element from the WebCache collection using the index of the cached item.

#### Syntax

```
object.Remove(Index As Variant)
```

The Remove method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type WebCache.
Index	Variant

#### Example

```
'In this example aWebCache represents a declared instance of the webcache class
'containing cached items

'removes the first cached item in the webcache collection

aWebCache.remove(0)
```

## RemoveAll

---

### RemoveAll

Removes all cached items from the WebCache Collection.

## Syntax

```
object.RemoveAll()
```

The RemoveAll method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type WebCache.

## Example

```
'In this example aWebCache represents a declared instance of the webcache class  
'containing cached items
```

```
'removes all cached item in the webcache collection
```

```
aWebCache.removeall
```

## Count

---

### Count

Returns the current number of Cached Items in the WebCache Collection - Read Only.

### Syntax

```
[value=]object.Count
```

The Count property syntax has the following parts:

Part	Description
object	A valid WebCache object
value	An Integer value.

### Data Type

Integer

### Example

```
'In this example aWebCache represents a declared instance of the webcache class  
'containing cached items
```

```
dim icount as integer  
icount= aWebCache.count
```

## WebCacheItem

---

### WebCacheItem

Name	Type	Description
Data	Property	Returns the data of the cached item.
Header	Property	Returns the header of the cached item.
Id	Property	Returns the cached items Id that is used by the ISAPI filter.
Persistence	Property	Determines when the cached items will be destroyed.
Timeout	Property	Determines the time in minutes that a cached item will remain in the cache.

### Data

---

## Data

Returns the data of the cached item - Read Only.

### Syntax

```
[value =]object.Data
```

The Data property syntax has the following parts:

Part	Description
object	A valid WebCacheItem object
value	A Variant value.

### Data Type

Variant

### Example

```
'This example demonstrates how to use the  
'Data property of the WebCacheItem class.  
'In the example "aWebCache" is a pre-existing  
'variable dimensioned as webcache and it  
'has been populated with a webcacheitem
```

```
Dim aWebCacheItem As New WebCacheItem  
Set aWebCacheItem = aWebCache.Item(0)  
debug.print aWebCacheItem.data
```

---

## Header

### Header

Returns the header of the cached item - Read Only.

### Syntax

```
[value =]object.Header
```

The Header property syntax has the following parts:

Part	Description
object	A valid WebCacheItem object
value	A String value.

### Data Type

String

### Example

```
'This example demonstrates how to use the  
'header property of the WebCacheItem class.  
'In the example "aWebCache" is a pre-existing  
'variable dimensioned as webcache and it  
'has been populated with a webcacheitem
```

```
Dim aWebCacheItem As New WebCacheItem  
Set aWebCacheItem = aWebCache.Item(0)  
  
debug.print aWebCacheItem.header
```

## Id

---

### Id

Returns the cached items Id that is used by the ISAPI filter - Read Only.

### Syntax

```
[value =]object.Id
```

The Id property syntax has the following parts:

Part	Description
object	A valid WebCacheItem object
value	A String value.

### Data Type

String

### Example

```
'This example demonstrates how to use the
'Id property of the WebCacheItem class.
'In the example "aWebCache" is a pre-existing
'variable dimensioned as webcache and it
'has been populated with a webcacheitem
```

```
Dim aWebCacheItem As New WebCacheItem
Set aWebCacheItem = aWebCache.Item(0)
```

```
debug.print aWebCacheItem.Id
```

## Persistence

---

### Persistence

Determines when the cached item will be destroyed - Read/Write.

### Syntax

```
object.Persistence [= value]
```

The Persistence property syntax has the following parts:

Part	Description
object	A valid WebCacheItem object
value	A PersistenceTypes value.

### Settings

The settings for value are:

Setting	Description
ddPermanent	1 - Cached item will stay alive forever. The item has to be destroyed using an explicit WebCache.Remove call.
ddTimeout	2 - Cached item will remain in the cache for a time period specified by the end user via the WebCacheItem's Timeout property. A possible usage scenario is setting the Timeout property to the SessionTimeout value under IIS.
ddAccessedOnce	3 - Cached item is destroyed immediately after the client accesses the data one time

### Data Type

## PersistenceTypes

### Example

```
'This example demonstrates how to use the
'Persistence property of the WebCachItem class.
'In the example "aWebCache" is a pre-existing
'variable dimensioned as webcache and it
'has been populated with a webcacheitem
```

```
Dim aWebCacheItem As New WebCacheItem
Set aWebCacheItem = aWebCache.Item(0)
```

```
aWebCacheItem.Persistence = 2
```

### Remarks

Default value = ddAccessedOnce

## TimeOut

---

### TimeOut

Determines the time in minutes that a cached item will remain in the cache - Read/Write.

Note: The Timeout property is only used if the persistence property of the WebCacheItem is set to 2 - ddTimeout.

### Syntax

```
object.TimeOut [= value]
```

The TimeOut property syntax has the following parts:

Part	Description
object	A valid WebCacheItem object
value	Long value.

### Data Type

Long

### Example

```
'This example demonstrates how to use the
'Timeout property of the WebCachItem class.
'In the example "aWebCache" is a pre-existing
'variable dimensioned as webcache and it
'has been populated with a webcacheitem
```

```
Dim aWebCacheItem As New WebCacheItem
Set aWebCacheItem = aWebCache.Item(0)
```

```
aWebCacheItem.Timeout = 2
```

### Remarks

Default value = 0

## WebCacheWorkerThread

---

### WebCacheWorkerThread

Name	Type	Description
AveragePerRequest	Property	Returns the average number of milliseconds per request.



NumberOfRequest                      Property Returns the number of requests that the thread has serviced.  
 ThreadId                                Property Returns the id of the WebCacheWorkerThread.  
 TotalTimeServicingRequest Property Returns the total time used servicing a request in milliseconds.

## TotalTimeServicingRequest

---

### TotalTimeServicingRequest

Returns the total time used servicing a request in milliseconds. The time waiting for a request is not included - Read Only.

#### Syntax

[value=] object.TotalTimeServicingRequest

The TotalTimeServicingRequest property syntax has the following parts:

Part	Description
object	A valid WebCacheWorkerThread object
value	A Long value.

#### Data Type

Long

#### Example

```
Dim numThreads As New WebCacheWorkerThreads
Dim aThread As New WebCacheWorkerThread

Set aThread = numThreads.Item(0)

Debug.Print aThread.ThreadID
Debug.Print aThread.TotalTimeServicingRequest
```

## ThreadId

---

### ThreadId

Returns the id of the WebCacheWorkerThread - Read Only.

#### Syntax

[value=] object.ThreadId

The ThreadId property syntax has the following parts:

Part	Description
object	A valid WebCacheWorkerThread object
value	An Integer value.

#### Data Type

Integer

#### Example

```
'This example prints out several properties
'for all of the workerthreads in the workerthreads
'collection. The sample adds the following properties to a
'standard vb listView control called lstThreads.
'ThreadId,AveragePerRequest,and NumberOfRequest.
```

```
Dim aItem As ListItem
```

```

Dim aThread As WebCacheWorkerThread
Dim aThreads As New WebCacheWorkerThreads
Dim nSize As Integer
Dim nIndex As Integer

nSize = aThreads.Count

For nIndex = 0 To nSize - 1
    Set aThread = aThreads.Item(nIndex)
    Set aItem = lstThreads.ListItems.Add(, , CStr(aThread.ThreadID))
    aItem.SubItems(1) = CStr(aThread.AveragePerRequest)
    aItem.SubItems(2) = CStr(aThread.NumberOfRequest)
Next nIndex

```

## NumberOfRequest

---

### NumberOfRequest

Returns the number of requests that the thread has serviced - Read Only.

#### Syntax

[value=] object.NumberOfRequest

The NumberOfRequest property syntax has the following parts:

Part	Description
object	A valid WebCacheWorkerThread object
value	A Long value.

#### Data Type

Long

#### Example

```

'This example prints out several properties
'for all of the workerthreads in the workerthreads
'collection. The sample adds the following properties to a
'standard vb listView control called lstThreads.
'ThreadId,AveragePerRequest,and NumberofRequest.

Dim aItem As ListItem
Dim aThread As WebCacheWorkerThread
Dim aThreads As New WebCacheWorkerThreads
Dim nSize As Integer
Dim nIndex As Integer

nSize = aThreads.Count

For nIndex = 0 To nSize - 1
    Set aThread = aThreads.Item(nIndex)
    Set aItem = lstThreads.ListItems.Add(, , CStr(aThread.ThreadID))
    aItem.SubItems(1) = CStr(aThread.AveragePerRequest)
    aItem.SubItems(2) = CStr(aThread.NumberOfRequest)
Next nIndex

```

## AveragePerRequest

---

### AveragePerRequest

Returns the average number of milliseconds per request - Read Only.

#### Syntax

[value=] object.AveragePerRequest

The AveragePerRequest property syntax has the following parts:

Part	Description
object	A valid WebCacheWorkerThread object
value	An Integer value.

## Data Type

Integer

## Example

```
'This example prints out several properties
'for all of the workerthreads in the workerthreads
'collection. The sample adds the following properties to a
'standard vb listView control called lstThreads.
'ThreadId,AveragePerRequest,and NumberOfRequest.
```

```
Dim aItem As ListItem
Dim aThread As WebCacheWorkerThread
Dim aThreads As New WebCacheWorkerThreads
Dim nSize As Integer
Dim nIndex As Integer

nSize = aThreads.Count

For nIndex = 0 To nSize - 1
    Set aThread = aThreads.Item(nIndex)
    Set aItem = lstThreads.ListItems.Add(, , CStr(aThread.ThreadID))
    aItem.SubItems(1) = CStr(aThread.AveragePerRequest)
    aItem.SubItems(2) = CStr(aThread.NumberOfRequest)
Next nIndex
```

## WebCacheWorkerThreads

---

### WebCacheWorkerThreads

Name	Type	Description
Item	Method	Returns the Thread object at the specified index.
Count	Property	Returns the number of WebCacheWorkerThreads in the collection.

## Count

---

### Count

Returns the current number of WebCacheWorkerThreads.

### Syntax

[value=] object.count

The Count property syntax has the following parts:

Part	Description
object	A valid WebCacheWorkerThreads object
value	An Integer value.

### Data Type

Integer

### Example

```
Dim numThreads As WebCacheWorkerThreads
Set numThreads = New WebCacheWorkerThreads

Debug.Print "workerthread count = " & numThreads.Count
```

## Item

---

### Item

Allows random access to individual nodes within the WebCacheWorkerThreads collection.

### Syntax

```
object.Item((Index As Variant))
```

The Item method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type WebCacheWorkerThreads.
Index	Variant

### Example

```
Dim numThreads As New WebCacheWorkerThreads
Dim aThread as New WebCacheWorkerThread

Set aThread = numThreads.Item(0)

Debug.Print aThread.Id
```

## Property List Control

---

### Property List Objects

PropList Control Object

PropNode Object

PropNodes Collection

### PropList Control

---

#### PropList

Name	Type	Description
AllowColumnResize	Property	Specifies whether the user is allowed to resize the property list columns.
BackColor	Property	Specifies the background color of the property list control.
BorderStyle	Property	Specifies the border style of the control.
Categorized	Property	Sets/returns if property list nodes are categorized or alphabetical.
Enabled	Property	Determines whether the property list control is enabled or disabled.
Font	Property	Specifies the font used to render text in the property list control.
ForeColor	Property	Specifies the foreground color of the property list.
hWnd	Property	Returns the property list window handle.
Properties	Property	Returns property nodes collection.

ShowDescription	Property	Sets/returns if property description pane is visible.
ShowObjectCombobox	Property	Sets/returns if object combobox is visible.
ShowReadOnlyProp	Property	Sets/returns weather readonly properties are shown.
ShowToolbar	Property	Sets/returns if toolbar is visible.
Sorted	Property	Determines whether the properties are sorted alphabetically in the list.
AddObject	Method	Adds an object reference to the property listbox and updates the combobox list.
Clear	Method	Removes all nodes from the property list.
Refresh	Method	Updates the propertylistbox with new values.
SelectObjects	Method	Sets the current selection. object can be a single COM object or an array of COM objects.
ButtonClick	Event	Fires when a button on ddPLButton property is clicked.
Error	Event	Fires when an internal error occurs in the property list control.
FetchData	Event	Fires when enum combobox dropdown is pressed.
FetchDataDescription	Event	Fires when combobox is updating its text or listbox.
ObjectChanged	Event	Fired when user selected a new object from the object combobox
PropertyChanged	Event	Fires when property value has been changed
PropertyValidate	Event	Fired before a value is stored in the property node when user makes a change to the value

## Sorted

### Sorted

Determines whether the properties are sorted alphabetically in the list.

### Syntax

object.Sorted [= value]

The Sorted property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Properties are sorted alphabetically.
False	Properties are listed in the order they were added.

### Data Type

Boolean

### Remarks

Default value = True

## ShowToolbar

### ShowToolbar

Sets/returns if toolbar is visible.

### Syntax

object.ShowToolbar [= value]

The ShowToolbar property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Displays the toolbar.
False	Hides the toolbar.

### Data Type

Boolean

### Remarks

Default

## ShowReadOnlyProp

---

### ShowReadOnlyProp

Sets/returns weather readonly properties are shown.

### Syntax

object.ShowReadOnlyProp [= value]

The ShowReadOnlyProp property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Displays the readonly properties.
False	Hides the readonly properties.

### Data Type

Boolean

### Remarks

Default value = True

## ShowObjectCombobox

---

### ShowObjectCombobox

Sets/returns if object combobox is visible.

### Syntax

object.ShowObjectCombobox [= value]

The ShowObjectCombobox property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Displays the objects combobox.
False	Hides the objects combobox.

### Data Type

Boolean

### Remarks

Default value = True

## ShowDescription

---

### ShowDescription

Sets/returns if property description pane is visible.

### Syntax

object.ShowDescription [= value]

The ShowDescription property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Description pane is visible.
False	Description pane is not visible.

### Data Type

Boolean

### Remarks

Default value = True

## Properties

---

### Properties

Returns property nodes collection.

### Syntax

Set value = object.Properties

The Properties property syntax has the following parts:

Part	Description
object	A valid object
value	A valid PropNodes collection.

### Data Type

IPropNodes

## hWnd

---

### hWnd

Returns the property list window handle.

### Syntax

value = object(hWnd)

The hWnd property syntax has the following parts:

Part	Description
object	A valid PropList object.
value	Returns the property list window handle.

### Data Type

OLE\_HANDLE

## ForeColor

---

### ForeColor

Specifies the foreground color of the property list.

### Syntax

object.ForeColor [= value]

The ForeColor property syntax has the following parts:

Part	Description
object	A valid PropList object.
value	A valid OLE_COLOR value.

### Data Type

OLE\_COLOR

### Remarks

Default value = vbWindowText

## Font

---

### Font

Specifies the font used to render text in the property list control.

### Syntax

object.Font [= value]

The Font property syntax has the following parts:



Part	Description
object	A valid PropList object
value	A valid Font object.

## Data Type

Font

## Enabled

### Enabled

Determines whether the property list control is enabled or disabled.

### Syntax

object.Enabled [= value]

The Enabled property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Property list control is enabled.
False	Property list control is disabled.

## Data Type

Boolean

## Remarks

Default value = True

## Categorized

### Categorized

Sets/returns if property list nodes are categorized or alphabetical.

### Syntax

object.Categorized [= value]

The Categorized property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Property list nodes are categorized in a treeview.
False	Property list nodes are listed alphabetically.

## Data Type

Boolean

## Remarks

Default value = True

## BorderStyle

---

### BorderStyle

Specifies the border style of the control.

### Syntax

object.BorderStyle [= value]

The BorderStyle property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A ddPLBorderStyle setting.

### Settings

The settings for value are:

Setting	Description
ddPLNone	0 - No border.
ddPLSunken	1 - Sunken border.

## Data Type

ddPLBorderStyle

## Remarks

Default value = ddPLSunken

## BackColor

---

### BackColor

Specifies the background color of the property list control.

### Syntax

object.BackColor [= value]

The BackColor property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A valid color value.

## Data Type

OLE\_COLOR

## Remarks

Default value = vbWindowBackColor

## AllowColumnResize

---

## AllowColumnResize

Specifies whether the user is allowed to resize the property list columns.

### Syntax

```
object.AllowColumnResize [= value]
```

The AllowColumnResize property syntax has the following parts:

Part	Description
object	A valid PropList object
value	A Boolean value.

### Settings

The settings for value are:

Setting	Description
True	Allows user to resize the property list columns.
False	Does not allow the user to size the columns.

### Data Type

Boolean

### Remarks

Default value = True

## SelectObjects

---

### SelectObjects

Sets the current selection. object can be a single COM object or an array of COM objects.

### Syntax

```
object_SelectObjects(selObject As Variant)
```

The SelectObjects method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
selObject	Variant - a single object or an array of objects.

### Example

```
' Select a single object to the property list
plist.SelectObjects Text1
```

```
'Select multiple objects (property list would
'aggregate common properties).
plist.SelectObjects Array(Text1, Text2, Text3)
```

## Refresh

---

### Refresh

Updates the propertylistbox with new values.

### Syntax

object\_Refresh()

The Refresh method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.

## Clear

---

### Clear

Removes all nodes from the property list.

#### Syntax

object\_Clear()

The Clear method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.

## AddObject

---

### AddObject

Adds an object reference to the property listbox and updates the combobox list.

#### Syntax

object\_AddObject(newObject As Object)

The AddObject method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
newObject	Object

#### Example

```
'Add an object to the property list
plist.AddObject Text1
plist.AddObject Text2
```

## PropertyValidate

---

### PropertyValidate

Fired before a value is stored in the property node when user makes a change to the value. Used to validate an entry.

#### Syntax

object\_PropertyValidate((property As IPropNode, newValue As Variant, Cancel As Boolean))

The PropertyValidate event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
property	PropNode - a reference to the current property.
newValue	Variant - new property value.
Cancel	Boolean - by ref parameter, allows you to cancel the change.

## ObjectChanged

---

### ObjectChanged

Fired when user selected a new object from the object combobox.

#### Syntax

```
object_ObjectChanged((newObject As Object))
```

The ObjectChanged event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
newObject	Object - a reference to the new selected object.

## PropertyChanged

---

### PropertyChanged

Fires when property value has been changed.

#### Syntax

```
object_PropertyChanged((property As IPropNode))
```

The PropertyChanged event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
property	PropNode - a reference to the changed property nodes.

## FetchDataDescription

---

### FetchDataDescription

Fires when comobox is updating its text or listbox. You can use this event to provide alternate description string for each enum value.

#### Syntax

```
object _FetchDataDescription((property As IPropNode, Value As Variant, Description As Variant))
```

The FetchDataDescription event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
property	PropNode
Value	Variant
Description	Variant

#### Example

```
' This example modifies the descriptions of all boolean properties to German
Private Sub PropList1_FetchDataDescription(ByVal property As DDPropertyListCtl.IPropNode, _
    ByVal Value As Variant, Description As Variant)
    If property.Type = ddPLBoolean Then
        If (Value = True) Then
            Description = "Ja"
        Else
            Description = "Nein"
        End If
    End If
```

End Sub

## FetchData

---

### FetchData

Fires when enum combobox dropdown is pressed. You can change the items in the combobox by using `node.ClearEnums` and `node.AddEnum` methods.

### Syntax

```
object_FetchData((property As IPropNode))
```

The FetchData event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
property	IPropNode

### Example

```
Private Sub pl_FetchData (ByVal property As DDPropertyListCtl.IPropNode)
    Select Case property.Name
        Case "State"
            property.ClearEnums
            property.AddEnum "AL", "Alabama"
            property.AddEnum "CA", "California"
            property.AddEnum "OH", "Ohio"
            property.AddEnum "NC", "North Carolina"
    End Select
End Sub
```

## Error

---

### Error

Fires when an internal error occurs in the property list control.

### Syntax

```
object_Error((Number As Integer, Description As ReturnString, Scode As Long,
Source As String, HelpFile As String, HelpContext As Long, CancelDisplay As ReturnBool))
```

The Error event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
Number	Integer - Error number.
Description	ReturnString - Brief description of the error.
Scode	Long - Result code.
Source	String - Error source.
HelpFile	String - Help file.
HelpContext	Long - Help context id.
CancelDisplay	ReturnBool - Boolean variable, used to suspend the built-in error message box.

### Example

```
' Handle PropertyList errors
Private Sub PropertyList1_Error(Number As Integer, Description As ReturnString,
    SCode As Long, Source As String, HelpFile As String, HelpContext As Long,
    CancelDisplay As Boolean)
```

```

'Display the error number and description to a form's status bar instead
'of an error message box
statusbar1.Panels(1).Text = "Error: " & Str(Number) & " - " & Description
CancelDisplay = True
End Sub

```

## ButtonClick

---

### ButtonClick

Fires when a button on ddPLButton property is clicked.

#### Syntax

```
object.ButtonClick(property As IPropNode)
```

The ButtonClick event syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropList.
property	IPropNode

#### Example

```

'Handle the border property with a custom dialog
Private Sub PropertyList1_ButtonClick(property as IPropNode)
    If property.Name = "Border" Then
        frmBorders.Show vbModal
    End If
End Sub

```

## PropNode Object

---

### PropNode

Name	Type	Description
AddEnum	Method	Adds a new enumeration value to the property
ClearEnums	Method	Clear all enumeration values for property
Category	Property	Sets/returns optional property category name.
Children	Property	Returns child property collection.
Description	Property	Sets/returns description for property.
Name	Property	Sets/returns property name.
Type	Property	Sets/returns UI type for property.
Value	Property	Sets/returns value of property

## AddEnum

---

### AddEnum

Adds a new enumeration value to the property.

#### Syntax

```
object.AddEnum(Value As Variant, Description As Variant)
```

The AddEnum method syntax has the following parts:

Part	Description
------	-------------

object	An expression evaluating to an object of type PropNode.
Value	Variant - value of the enum.
Description	Variant - description of the enum.

### Example

```
Private Sub pl_FetchData(ByVal property As DDPropertyListCtl.IPropNode)
    Select Case property.Name
        Case "State"
            property.ClearEnums
            property.AddEnum "AL", "Alabama"
            property.AddEnum "CA", "California"
            property.AddEnum "OH", "Ohio"
            property.AddEnum "NC", "North Carolina"
    End Select
End Sub
```

## ClearEnums

---

### ClearEnums

Clear all enumeration values for property.

#### Syntax

object.ClearEnums()

The ClearEnums method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropNode.

### Example

```
Private Sub pl_FetchData(ByVal property As DDPropertyListCtl.IPropNode)
    Select Case property.Name
        Case "State"
            property.ClearEnums
            property.AddEnum "AL", "Alabama"
            property.AddEnum "CA", "California"
            property.AddEnum "OH", "Ohio"
            property.AddEnum "NC", "North Carolina"
    End Select
End Sub
```

## Category

---

### Category

Sets/returns optional property category name.

#### Syntax

object.Category [= value]

The Category property syntax has the following parts:

Part	Description
object	A valid PropNode object
value	A String value.

#### Data Type

String



## Children

---

### Children

Returns child property collection.

### Syntax

object.Children [= value]

The Children property syntax has the following parts:

Part	Description
object	A valid PropNode object
value	A PropNodes Collection.

### Data Type

IPropNodes

### Example

```
'Create a complex property Address with child nodes.
Set nod = New PropNode
nod.Category = "Address"
nod.Name = "Telephone"
nod.Type = ddPLLabel
Set subNod = New PropNode
subNod.Category = "Address"
subNod.Name = "Home"
subNod.Type = ddPLString
nod.Children.Add subNod
Set subNod = New PropNode
subNod.Category = "Address"
subNod.Name = "Business"
subNod.Type = ddPLString
nod.Children.Add subNod
pl.Properties.Add nod
```

## Description

---

### Description

Sets/returns description for property.

### Syntax

object.Description [= value]

The Description property syntax has the following parts:

Part	Description
object	A valid PropNode object
value	A String value.

### Data Type

String

### Name

---

Name

Sets/returns property name.

### Syntax

object.Name [= value]

### Values

The Name property syntax has the following parts:

Part	Description
object	A valid PropNode object
value	A String value.

### Data Type

String

## Type

---

### Type

Sets/returns UI type for property.

### Syntax

object.Type [= value]

The Type property syntax has the following parts:

Part	Description
object	A valid PropNode object
value	A ddPLNodeType setting.

### Settings

The settings for value are:

Setting	Description
ddPLString	0 - A string property.
ddPLLabel	1 - A static label.
ddPLEnum	2 - An enumerated property editor.
ddPLBoolean	3 - A Boolean property editor.
ddPLColor	4 - A color property editor.
ddPLStringCombo	5 - A string editor with a combobox.
ddPLPicture	6 - A picture property editor.
ddPLFont	7 - A font property editor.
ddPLButton	16 - Adds a custom button to the property editor, can be combined with any of the other types.

### Data Type

ddPLNodeType

## Value

---

### Value

Sets/returns value of property. Call the refresh method to update the property listbox with the new value.

### Syntax

object.Value [= value]

### Values

The Value property syntax has the following parts:

Part	Description
object	A valid PropNode object
value	A Variant value.

### Data Type

Variant

## PropNodes Collection

---

### PropNodes

Name	Type	Description
Add	Method	Adds the specified node object to the collection.
Count	Method	Returns the number of property nodes in the collection.
Item	Method	Returns the property node object at the specified index.
Remove	Method	Removes a node from the collection at the specified index

## Remove

---

### Remove

Removes a node from the collection at the specified index.

### Syntax

object.Remove(Index As Variant)

The Remove method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropNodes.
Index	Variant - Index of the node to be removed from the collection.

## Item

---

### Item

Returns the property node object at the specified index.

### Syntax

object.Item(Index As Variant)

The Item method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropNodes.

Index    Variant

## Count

---

### Count

Returns the number of property nodes in the collection.

### Syntax

```
object.Count()
```

The Count method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropNodes.

## Add

---

### Add

Adds the specified node object to the collection.

### Syntax

```
object.Add(property As PropNode)
```

The Add method syntax has the following parts:

Part	Description
object	An expression evaluating to an object of type PropNodes.
property	PropNode object to be added.

### Example

```
Set nod = New PropNode  
nod.Category = "Address"  
nod.Name = "State"  
nod.Type = ddPLEnum  
pl.Properties.Add nod
```