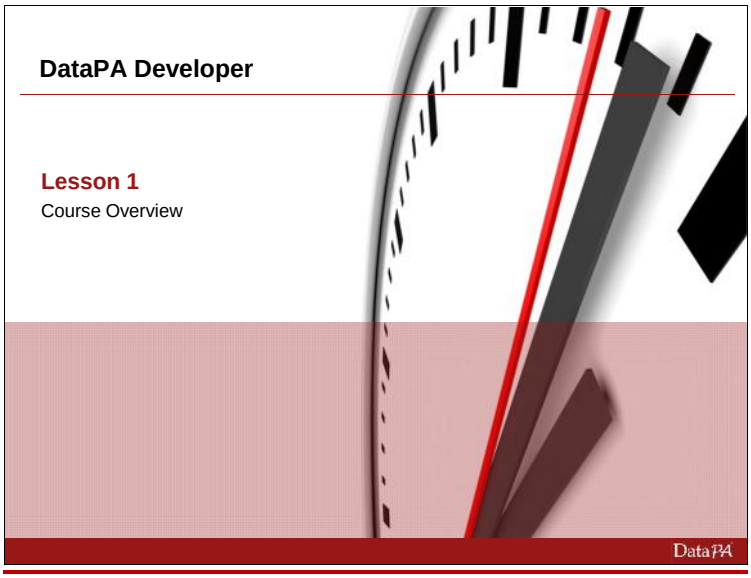
DataPA
OpenAnalytics

PA

DataPA Development

for Openedge Developers

Part 1



Course Overview

Introduction

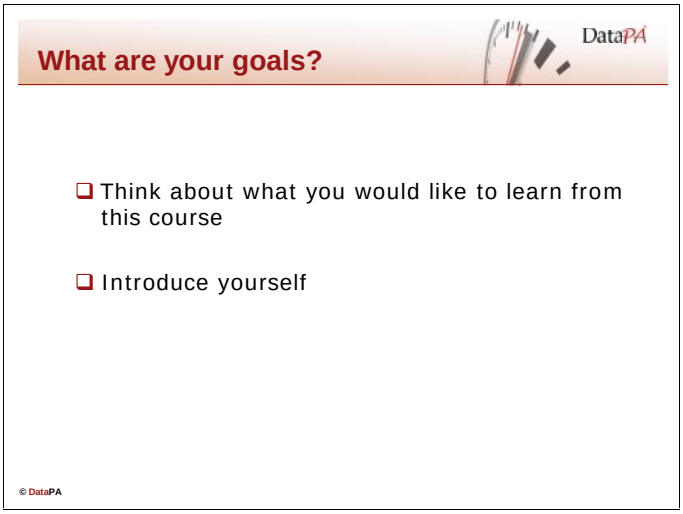
This course covers the skills and knowledge required to develop applications which use DataPA OpenAnalytics to enhance their functionality.

Audience

The course is intended for developers who will either develop or maintain applications which use DataPA OpenAnalytics. The course will not cover the administration of DataPA OpenAnalytics, the Progress AppServer's or the use of the product itself. These topics are covered in other courses.

Prerequisites

Students should be familiar with one of the development tools which are covered in this course which are the Progress AppBuilder, Microsoft Visual Studio or OpenEdge Architect.



Student Goals

Please take a few moments to document your own goals for this course. What will you need to know and/or produce when you return to work?

What are the three things you most want to know about DataPA?

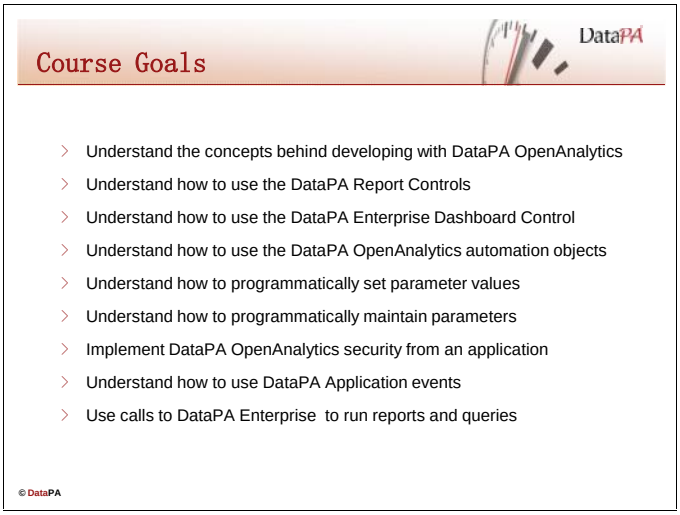
-
-
-

Please introduce yourself by answering the following questions

1. Your Name and Job

2. Your experience with Progress and DataPA OpenAnalytics

3. What you would like to learn from this course



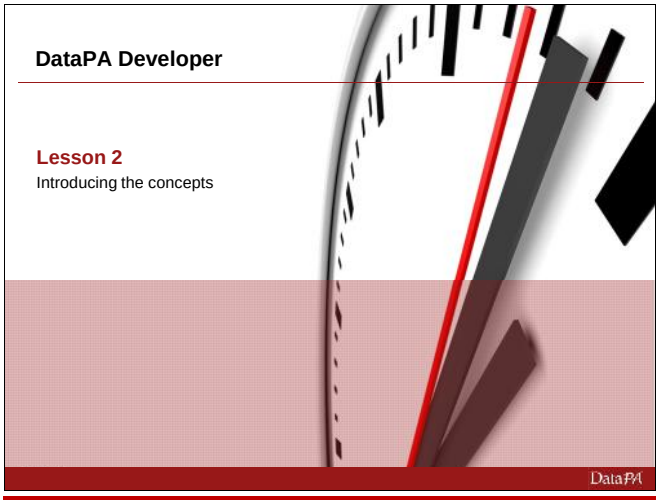
Course Goals

When you complete this course you should be able to:

- Understand the concepts behind developing with DataPA OpenAnalytics
- Understand how to use the DataPA Report Controls
- Understand how to use the DataPA Enterprise Dashboard Control
- Understand how to use the DataPA OpenAnalytics automation objects
- Understand how to programmatically set parameter values
- Understand how to programmatically maintain parameters
- Implement DataPA OpenAnalytics security from an application
- Understand how to use DataPA Application events
- Use calls to DataPA Enterprise to run reports and queries

Lesson Overview

Lesson	What is covers
Lesson 1: Course Overview	Introductory material about this course.
Lesson 2: Introducing the concepts	An introduction to the concepts used in the course and a look at some of the best practise recommendations.
Lesson 3: DataPA Reports Control	A detailed look at using the DataPA Reports Control to design, view, run, print, export, open, save and close reports from your application.
Lesson 4: DataPA Enterprise Dashboard Control	A detailed look at using the DataPA Enterprise Dashboard Control to design, view, refresh data and save settings from your application.
Lesson 5: DataPA Application Object	A detailed look at using the DataPA application object to control the queries behind the reports you are using from your application.
Lesson 6: Managing Query Parameters	A detailed look at how to manipulate DataPA Query parameters from your application to control the data shown on the reports.
Lesson 7: Setting Security Information	A look at how to set the security context for a user of your application.
Lesson 8: Using DataPA Application Events	A look at how you can respond to DataPA Application events from your application.
Lesson 9: Using DataPA Enterprise	A look at how use DataPA Enterprise from your application.



Introducing the concepts

Introduction

DataPA Developer allows developers to embed the full functionality of DataPA into their client applications and this lesson will lay out some of key concepts behind using DataPA in this way together with some of the best practice recommendations for developing your application with DataPA.

Learning Objectives

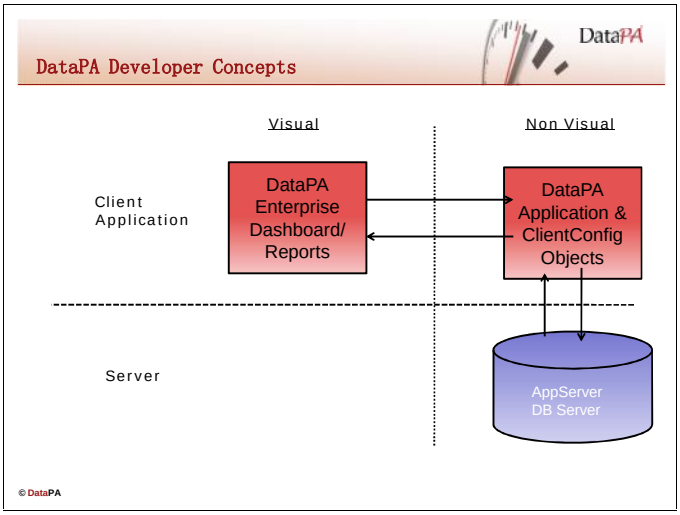
When you complete this lesson you should be able to:

- Explain DataPA concepts behind using DataPA in your application.
- Understand the best practice recommendations and why they are important.

Prerequisites

Before you begin this lesson you should be able to:

- Develop an application in your chosen development environment.



DataPA Developer Concepts

In order to define understand how DataPA OpenAnalytics works it is important to know the principle components which DataPA OpenAnalytics uses to allow dashboards, reports and queries to be run from client applications.

DataPA Enterprise Dashboard Object

The DataPA Enterprise Dashboard Control is a visual object which can be dropped onto a form in a client application. This Control allows the developer to embed both dashboard design and preview functionality into their application. It has an API (Application Programming Interface) available for the developer to use to control many aspects of the behaviour of the control.

For applications developed using the Progress AppBuilder the DataPA Enterprise Dashboard Control is an ActiveX control that can be added to Progress windows in the same way as other OCX controls.

For applications developed using Visual Studio or OpenEdge Architect the DataPA Enterprise Dashboard Control is a .NET control that can be added to a .NET form or an ABL Form in OpenEdge Architect.

The OCX version of the DataPA Enterprise Dashboard control has the same API the .NET version, however due to the restrictions of the older COM interface, some of the properties or methods cannot be used in the COM interface. For these properties or methods, alternative COM versions of the properties or methods are available.

DataPA Reports Control

The DataPA Report Control is a visual object which can be dropped onto a form in a client application. This Control allows the developer to embed both report design and preview functionality into their application. It has an API (Application Programming Interface) available for the developer to use to control many aspects of the behaviour of the control.

For applications developed using the Progress AppBuilder the DataPA Reports Control is an ActiveX control that can be added to Progress windows in the same way as other OCX controls.

For applications developed using Visual Studio or OpenEdge Architect the DataPA Reports Control is a .NET control that can be added to a .NET form or an ABL Form in OpenEdge Architect.

The OCX version of the DataPA Reports control has the same API the .NET version.

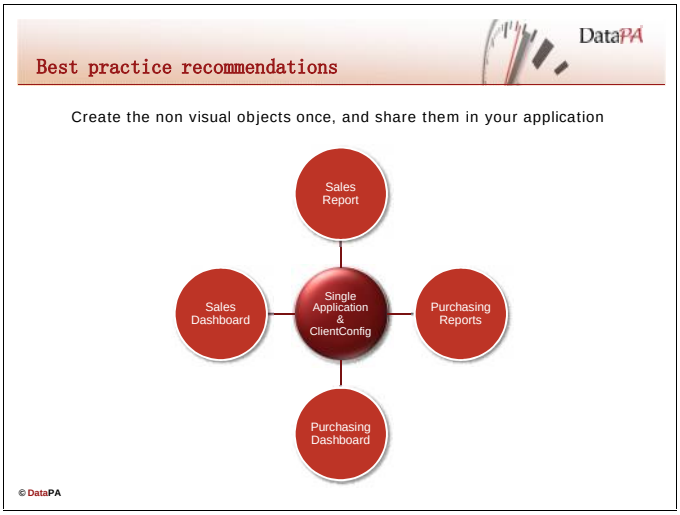
DataPA Application Object

The DataPA Application Object is a non visual class which has an extensive API that allows the developer to control many aspects of the functionality of DataPA OpenAnalytics. The DataPA Reports Control and DataPA Enterprise Dashboard both use the DataPA Application Object to perform all the data related tasks involved in rendered a report or dashboard within your client application.

The DataPA Application Object communicates directly with the Progress AppServer in order to retrieve data for the report or dashboard.

DataPA ClientConfig Object

The DataPA ClientConfig object is a non visual class which has an extensive API that allows the developer to control many aspects of the functionality of DataPA OpenAnalytics. The DataPA Reports Control and DataPA Enterprise Dashboard both use the DataPA ClientConfig object to perform all the security related tasks involved in rendered a report or dashboard within your client application.



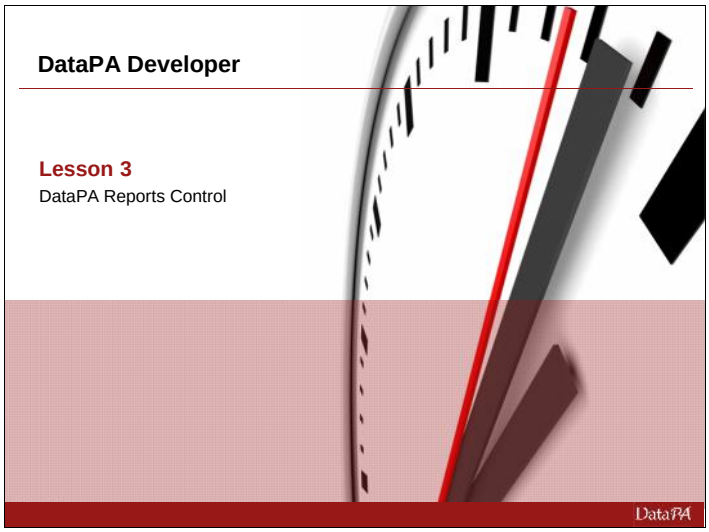
Best practice guidelines for developing with DataPA

In order to get the most out of developing with DataPA some best practice guidelines have been identified. Putting there guidelines into practice will be looked at in more detail in later chapters of this course.

Create the non visual objects once, and share them in your application

In order to avoid creation and destruction of multiple DataPA Application and ClientConfig objects it is recommended that they are created once, when first required by the client application. After this they should be keep in memory so that is available quickly to service any other requests which occur during the lifetime of the client application.

The single Application and ClientConfig objects can be used by forms which use the DataPA Reports or DataPA Enterprise Dashboard controls. This helps ensure better response times for the end users for the client application.



DataPA Reports Control

Introduction

The DataPA report controls allow the DataPA report designer and preview window to be embedded into any application to provide report functionality. This chapter concentrates on using the report designer controls with Progress GUI, Visual Studio and OpenEdge Architect clients.

Learning Objectives

When you complete this lesson you should be able to:

- Configure your development environment to use the DataPA report controls.
- Add the DataPA Reports control to a form.
- Use the controls properties and methods to control its behaviour.
- Respond to events triggered by the DataPA report controls.

Prerequisites

Before you begin this lesson you should be able to:

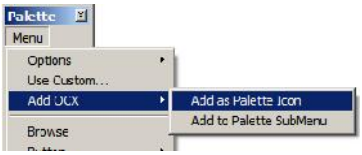
- Create a GUI application using Progress
- Setup and configure DataPA for an AppServer application

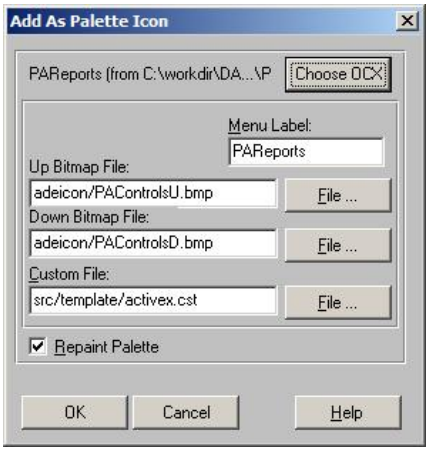
Configuring Progress AppBuilder

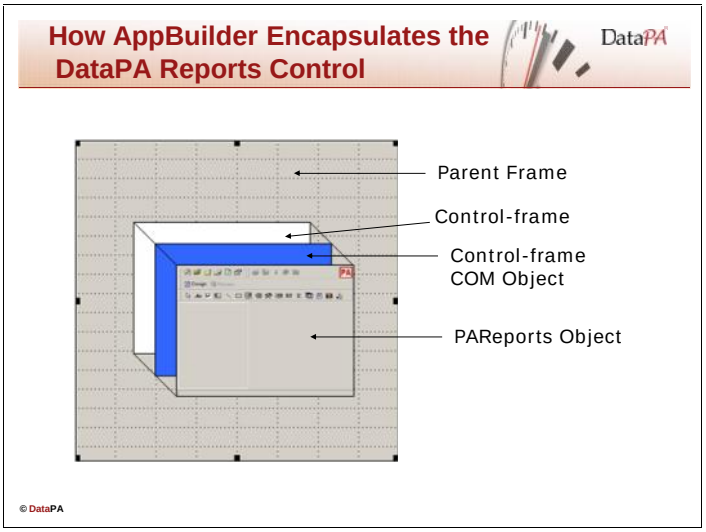
To provide quick and simple access to the DataPA Report Control in the AppBuilder, we can add it to the AppBuilder Object Palette.

Follow these steps to add the DataPA Report Control to the AppBuilder.

- Start a copy of the the Progress AppBuilder
- From the AppBuilder Object Palette menu, select Add OCX
- Add as Palette Icon
- Press the Choose OCX button
- Select DataPAReportsControl.PAReports from the selection list
- Press OK to close the list of available OCX objects
- Press OK to close the Add As Palette Icon dialog box.







How the AppBuilder encapsulates the DataPA Reports Control

The DataPA Reports Control is an ActiveX control or OCX. To comply with the COM object standard, an ActiveX control instance must be placed in a *control container* that handles events and specific user-interface functionality for the control. Progress supports this standard with the control-frame widget, the container, called a control-frame COM object and the control itself, in our case a PAReports object.

Control-frame Widget

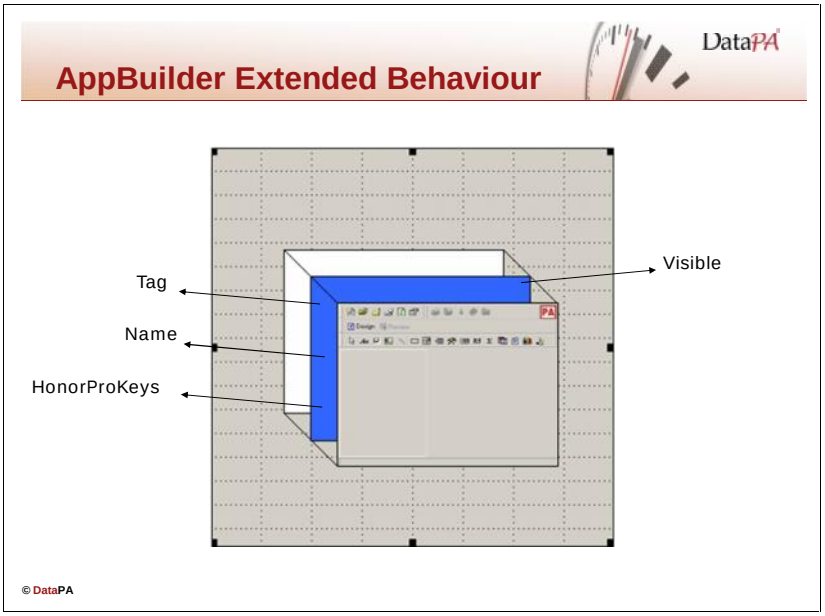
The control frame widget is a specialised Progress frame that establishes the relationship between a PAReports control, the other Progress widgets in the user interface, and the 4GL. As such the control-frame gives access to properties that interact with the other 4GL widgets, (like ROW and TAB-POSITION), and events that interact with other widgets (like TAB and LEAVE) or have specific relevance in the 4GL (like GO and END-ERROR).

The control-frame has the following attributes:

Widget Attribute	COM Object Property
HEIGHT-PIXELS, HEIGHT[-CHARS]	Height
NAME	Name
WIDTH-PIXELS, WIDTH[-CHARS]	Width
X, COLUMN	Left
Y, ROW	Top

Control-frame COM Object

The control-frame COM object is the actual PAReports control container. It provides the initial point of access to the PAReports control from the 4GL. Through the COM object you can access the component handle of the PAReports control to directly access its properties and methods.



AppBuilder Extended Behaviour

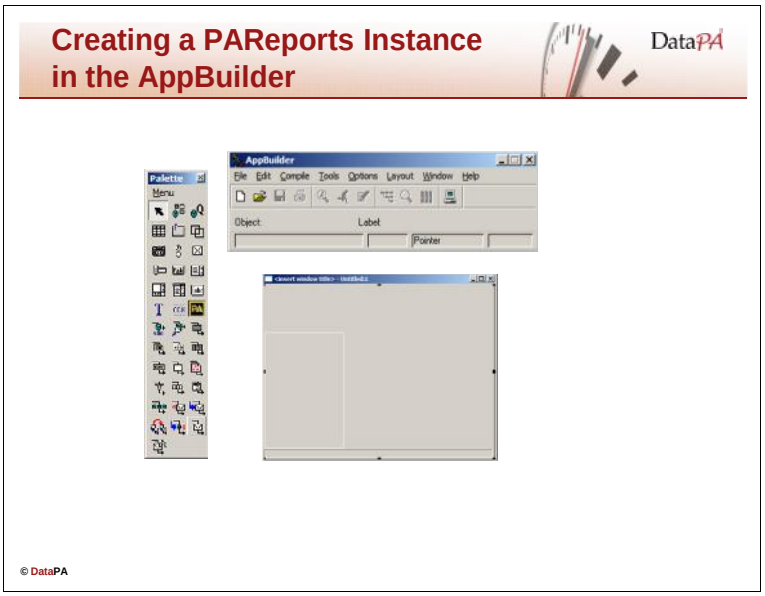
In addition to providing an initial point of access for the PAReports control, the control-frame COM object acts as a wrapper to the PAReports com object providing extended functionality specific to the Progress environment. Even though the control-frame COM object provides this extended functionality, and the functionality is completely unknown to the PAReports com object, the extended methods, properties and events appear as though part of the PAReports object itself and are accessed through the same com object handle (com object handles will be discussed later in this lesson).

Extended Properties

The table below lists the extended properties available for the PAReports control:

Property Name	Type	Definition
HonorProKeys	logical	Default is TRUE. Determines who processes the GO, ENDKEY, HELP, and TAB keys: Progress, or the PAReports control. If the property is TRUE, Progress intercepts these keys and processes them as normal Progress key events. If the property is FALSE, the keystrokes are sent to the PAReports control for processing.
HonorReturnKey	logical	Default is FALSE. If the property is TRUE, Progress intercepts the key and processes it as a normal Progress RETURN key event. If the property is FALSE, which is the default, the keystroke is sent to the ActiveX control for processing.
Name	string	The Name property contains the name of the control. The name is important because it identifies the control. You can use the control's name to get a COM-HANDLE to the control (for example, ASSIGN chPAReports = chCtrlFrame:PAReports, where PAReports is the control's name and chCtrlFrame is the control frame handle). The control name associates event handlers with a control.

Property Name	Type	Definition
Parent	COM-Handle	The Parent property is the com-handle of the container in which the control resides. This property is set internally by Progress.
Tag	string	The Tag property is a user property that allows the user to store an arbitrary string value and retrieve it later. Progress does not use this property internally, and it is intended to give the user a way of storing application specific information with the control. This property is initialized to an empty string.
Visible	Logical	The Visible property determines and indicates whether a PAReports control is currently displayed. The Visible property is distinct from, but influenced by, the Visible and Hidden attributes of the Control-Frame widget. The Visible property will appear in the Property Editor and can be set at design time. It defaults to TRUE. The value set in the Property Editor determines whether the OCX is initially displayed when the program is run, but can be overridden by the value of the Control-Frame widget's Hidden attribute.



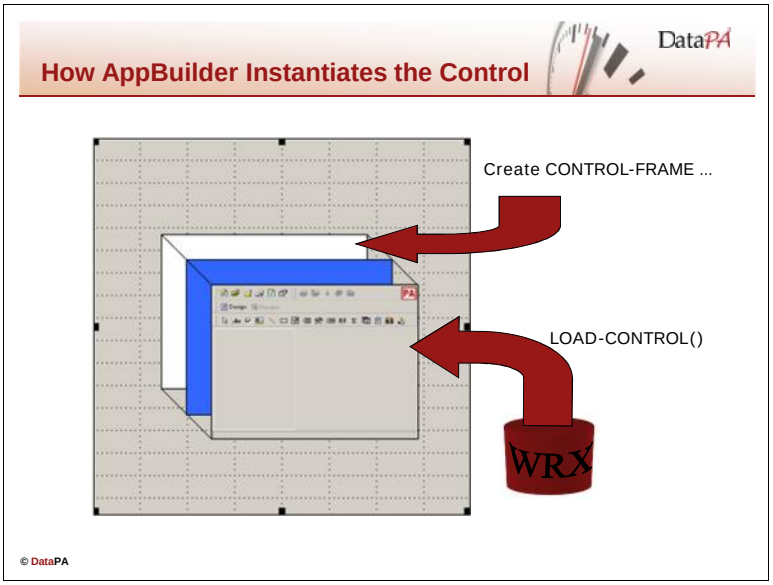
Creating a PAReports Instance in the AppBuilder

You must use the AppBuilder to add a PAReports control to a Progress user interface. Follow these steps in the AppBuilder to add a PAReports control:

- Open a container object, such as a Window, Dialog or SmartWindow.
- Choose the PAReports ActiveX control from the object palette.
- Click on the container object to place the PAReports control onto the container.
- Resize the PAReports control as required.

If you save your work at this point, the AppBuilder generates the following data and code for your application:

- The definition for the default instance of the PAReports control in a binary (.wrx) file. This includes the initial definition provided by DataPA.
- The default 4GL code in your .w file to instantiate and orient the PAReports control in the user interface at runtime.



How the AppBuilder Instantiates the Control

At runtime, Progress uses AppBuilder-generated code to instantiate the PAReports control. This code uses the CREATE Widget statement to realize a control-frame widget, initializing the control-frame with the name specified at design time. It then invokes the LoadControls( ) method on the control-frame COM object to instantiate the PAReports control, loading all design time definitions from the .wrx file.

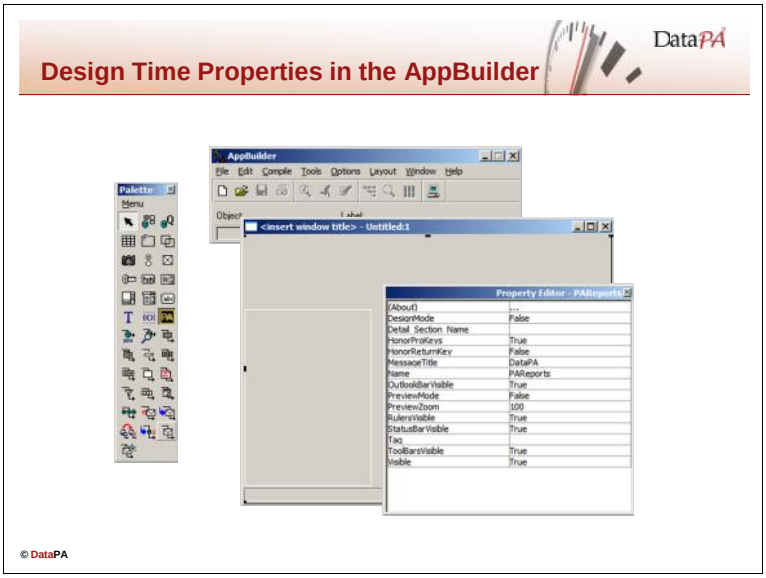
Example

The following AppBuilder-generated code creates the control-frame in our example application:

```
CREATE CONTROL-FRAME CtrlFrame ASSIGN
       FRAME           = FRAME DEFAULT-FRAME:HANDLE
       ROW             = 1
       COLUMN          = 1
       HEIGHT          = 21.67
       WIDTH           = 137
       WIDGET-ID       = 2
       HIDDEN          = no
       SENSITIVE       = yes.
      CtrlFrame:NAME = "CtrlFrame":U .
```

The following code calls the LoadControls method to load the control into the control-frame COM object.

```
DEFINE VARIABLE UIB_S    AS LOGICAL    NO-UNDO.
DEFINE VARIABLE OCXFile  AS CHARACTER  NO-UNDO.

OCXFile = SEARCH( "{&FILE-NAME}.wrx":U ).
IF OCXFile = ? THEN
  OCXFile = SEARCH(SUBSTRING(THIS-PROCEDURE:FILE-NAME, 1,
                    R-INDEX(THIS-PROCEDURE:FILE-NAME, ".":U), "CHARACTER":U)
+ "wrx":U).

IF OCXFile <> ? THEN
DO:
  ASSIGN chCtrlFrame = CtrlFrame:COM-HANDLE
         UIB_S = chCtrlFrame:LoadControls( OCXFile, "CtrlFrame":U).
  RUN initialize-controls IN THIS-PROCEDURE NO-ERROR.
END.
```



Design Time Properties in the AppBuilder

To customise the definition for a PAReports control, you must change the values of control properties in the AppBuilder at design time. The PAReports control supports properties that can be changed at both design time and run time, and some properties that can only be changed at runtime (*runtime properties*).

Setting Design Time Properties

The AppBuilder provides access to all available design time properties using the OCX Property Editor window. This window contains all design time properties, including the extended properties that Progress adds. Follow these steps in the AppBuilder to open the OCX Property Editor:

- Open a container object that contains a PAReports object.
- Double-click on the PAReports object.

The properties available in the OCX Property Editor window are described in the table below:

Property Name	Type	Definition
DesignMode	logical	Determines whether the PAReports module shows the report design interface.
HonorProKeys	logical	Default is TRUE. Determines who processes the GO, ENDKEY, HELP, and TAB keys: Progress, or the PAReports control. If the property is TRUE, Progress intercepts these keys and processes them as normal Progress key events. If the property is FALSE, the keystrokes are sent to the PAReports control for processing.
HonorReturnKey	logical	Default is FALSE. If the property is TRUE, Progress intercepts the key and processes it as a normal Progress RETURN key event. If the property is FALSE, which is the default, the keystroke is sent to the ActiveX control for processing.

Property Name	Type	Definition
MessageTitle	string	Sets the title that will appear on any messages displayed by the PAReports object. The default is DataPA.
Name	string	The Name property contains the name of the control. The name is important because it identifies the control. You can use the control's name to get a COM-HANDLE to the control (for example, ASSIGN chPAReports = chCtrlFrame:PAReports, where PAReports is the control's name and chCtrlFrame is the control frame handle). The control name associates event handlers with a control.
OutlookbarVisible	Logical	Determines whether the PAReports outlookbar is visible. Default is TRUE.
PreviewMode	logical	Determines whether the PAReports module shows the report preview interface.
PreviewZoom	Integer	The PreviewZoom property controls the extent to which the report preview window zooms into the image of the report.
RulersVisible	Logical	The RulersVisible property controls whether or not the rulers are visible in the preview pane of the PAReports control.
StatusBarVisible	Logical	The StatusBarVisible property controls whether or not the status bar is visible at the base of the PAReports control.
Parent	COM-Handle	The Parent property is the com-handle of the container in which the control resides. This property is set internally by Progress.
Tag	string	The Tag property is a user property that allows the user to store an arbitrary string value and retrieve it later. Progress does not use this property internally, and it is intended to give the user a way of storing application specific information with the control. This property is initialized to an empty string.
ToolBarsVisible	logical	The ToolBarsVisible property controls whether or not the tool bars are visible at the top of the PAReports control.
Visible	Logical	The Visible property determines and indicates whether a PAReports control is currently displayed. The Visible property is distinct from, but influenced by, the Visible and Hidden attributes of the Control-Frame widget. The Visible property will appear in the Property Editor and can be set at design time. It defaults to TRUE. The value set in the Property Editor determines whether the OCX is initially displayed when the program is run, but can be overridden by the value of the Control-Frame widget's Hidden attribute.

Configuring Visual Studio

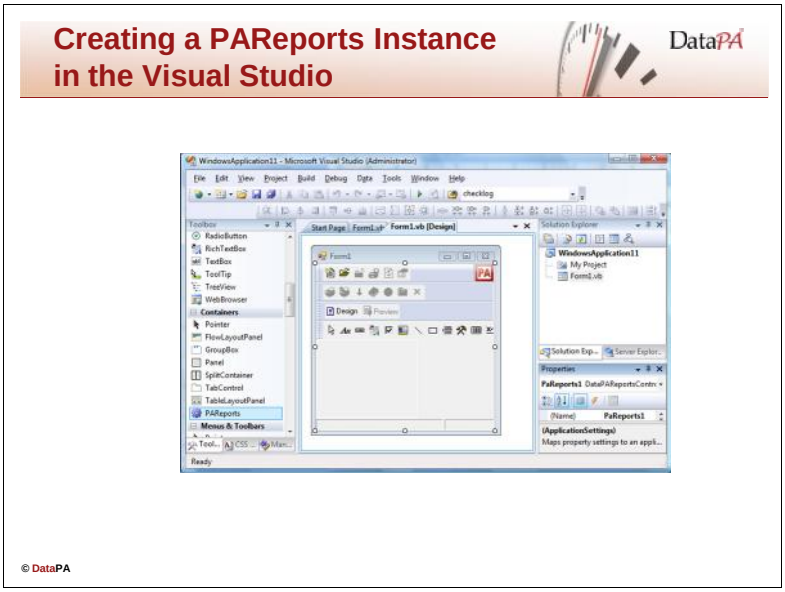
To provide quick and simple access to the DataPA Report Control in the Visual Studio, we can add it to Toolbox.

Follow these steps to add the DataPA Report Control to the Toolbox.

- Start a copy of the Microsoft Visual Studio
- Create a new Windows Forms application
- Right click on the Toolbox and select Choose Items.
- Then check the PAReportsControl item from the list on the .NET Framework Components tab and click OK.

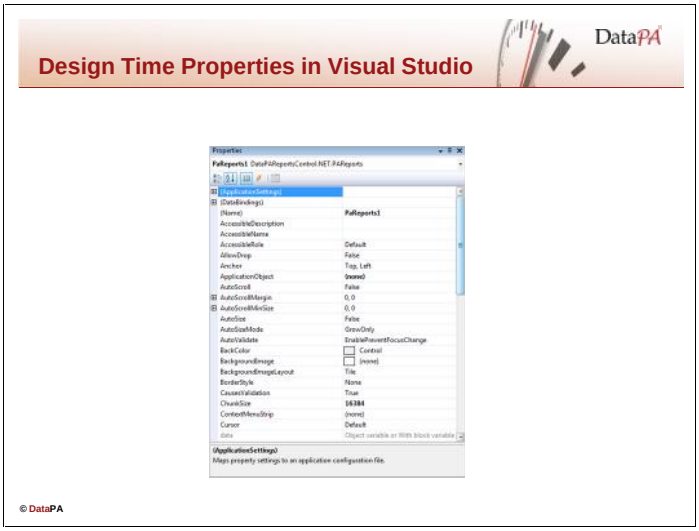
Using the DataPA Reports control in the Visual Studio

When using DataPA with Visual Studio it is best to use the DataPA Reports .NET control provided. This is a .NET control that Visual Studio supports natively. As well as referencing the .NET control a reference should be added to the Interop assembly for the DataPA Application object.



Creating a PAReports Instance in the Visual Studio

To add an instance DataPA Reports control to your form select the PAControls item that was added to the Toolbox in Visual Studio earlier in this lesson and draw the control onto the form.



Design Time Properties in the Visual Studio

To customise the definition for a PAReports control, you must change the values of control properties in the Visual Studio at design time. The PAReports control supports properties that can be changed at both design time and run time, and some properties that can only be changed at runtime (*runtime properties*).

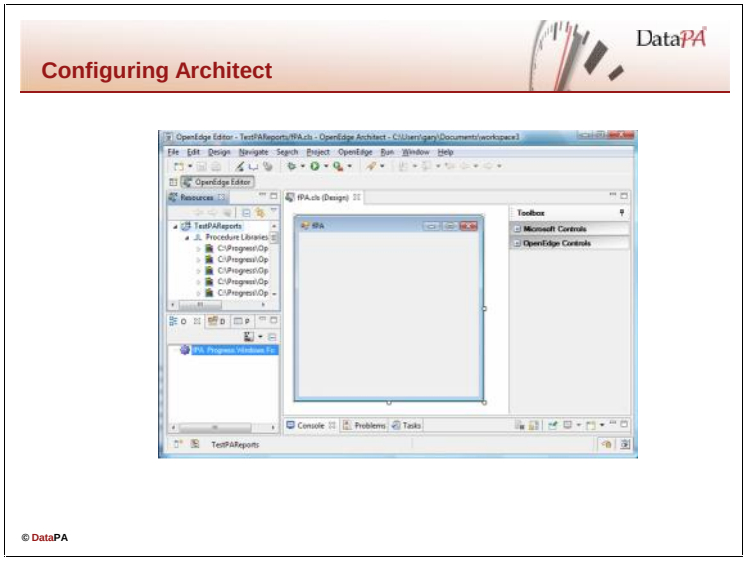
Setting Design Time Properties

Visual Studio provides access to all available design time properties using the Properties Toolbox. This window contains all design time properties, including the extended properties that Visual Studio adds.

The properties available in the Property Toolbox window are described in the table below:

Property Name	Type	Definition
DesignMode	logical	Determines whether the PAReports module shows the report design interface.
MessageTitle	string	Sets the title that will appear on any messages displayed by the PAReports object. The default is DataPA.
Name	string	The Name property contains the name of the control. The name is important because it identifies the control. You can use the control's name to access the properties and methods of the control.
OutlookbarVisible	Logical	Determines whether the PAReports outlookbar is visible. Default is TRUE.
PreviewMode	logical	Determines whether the PAReports module shows the report preview interface.
PreviewZoom	Integer	The PreviewZoom property controls the extent to which the report preview window zooms into the image of the report.

Property Name	Type	Definition
RulersVisible	Logical	The RulersVisible property controls whether or not the rulers are visible in the preview pane of the PAReports control.
StatusBarVisible	Logical	The StatusBarVisible property controls whether or not the status bar is visible at the base of the PAReports control.
Parent	Handle	The Parent property is the handle of the container in which the control resides. This property is set internally by Visual Studio.
Tag	string	The Tag property is a user property that allows the user to store an arbitrary string value and retrieve it later. Progress does not use this property internally, and it is intended to give the user a way of storing application specific information with the control. This property is initialized to an empty string.
ToolBarsVisible	logical	The ToolBarsVisible property controls whether or not the tool bars are visible at the top of the PAReports control.
Visible	Logical	The Visible property determines and indicates whether a PAReports control is currently displayed. The Visible property will appear in the Property Toolbox and can be set at design time. It defaults to TRUE. The value set in the Property Editor determines whether the control is initially displayed when the project is run.

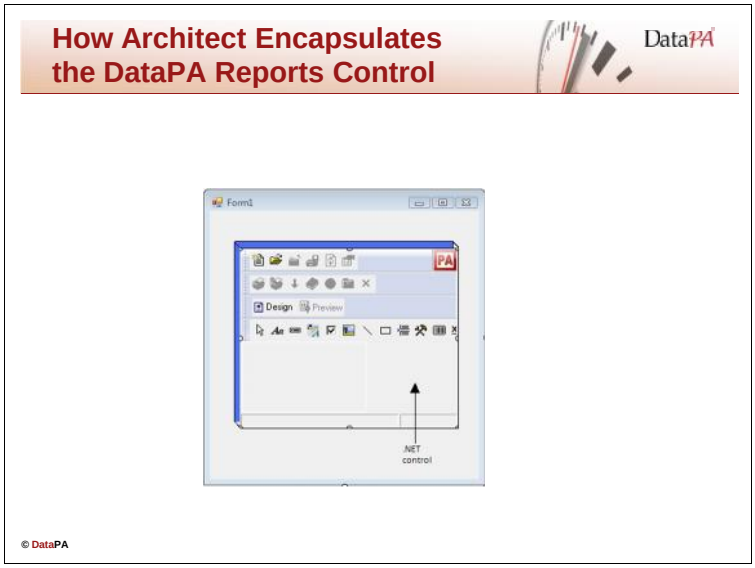


Configuring Architect

To provide quick and simple access to the DataPA Report Control in the Architect, we can add it to one of the control groups in the Toolbox.

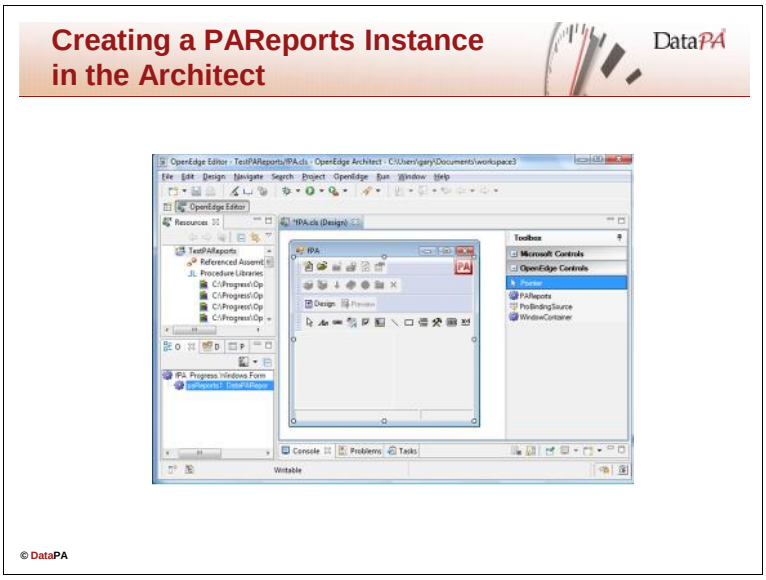
Follow these steps to add the DataPA Report Control to the Toolbox.

- Start a copy of the OpenEdge Architect
- Create a new OpenEdge Project
- In the Resources Treeview select the newly created project and right click
- Select New and then ABL Form
- Right click on the toolbox and select Add Controls
- Click on the Global Assemblies tab
- Check the PAReportsControl assembly from the list and select OK
- The PAReports controls should now be available in the Toolbox.



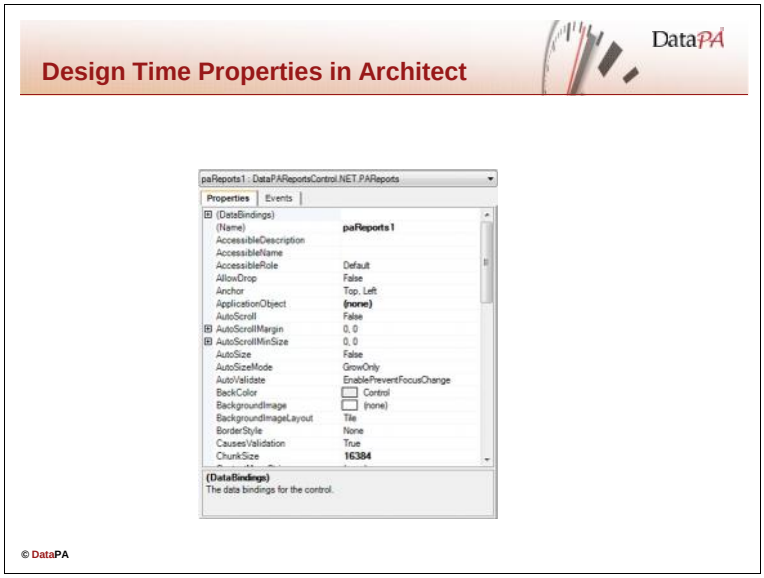
Using the DataPA Reports control in OpenEdge Architect

When using DataPA with Architect it is best to use the DataPA Reports .NET control provided. This is a .NET control that Architect supports natively.



Creating a PAReports Instance in the Architect

To add an instance DataPA Reports control to your form select the PAControls item that was added to the Toolbox in Architect earlier in this lesson and draw the control onto the form.



Design Time Properties in Architect

To customise the definition for a PAReports control, you must change the values of control properties in the Architect at design time. The PAReports control supports properties that can be changed at both design time and run time, and some properties that can only be changed at runtime (*runtime properties*).

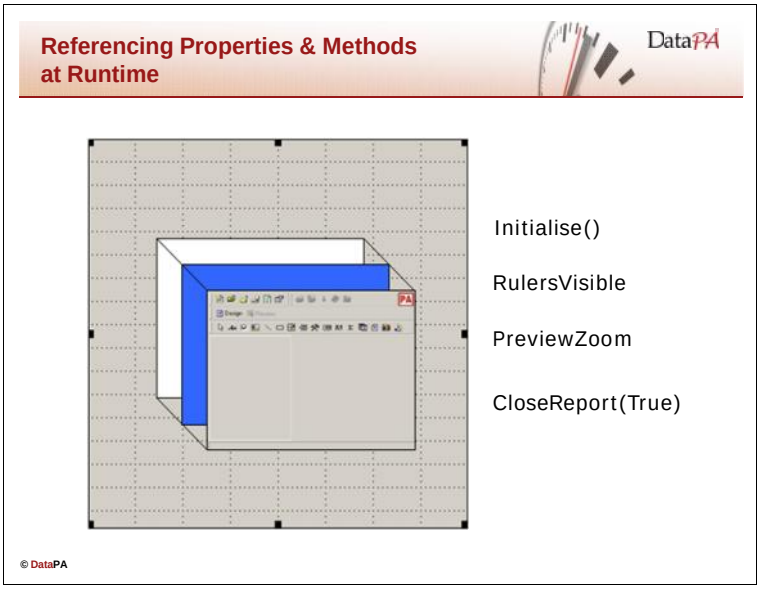
Setting Design Time Properties

Architect provides access to all available design time properties using the Toolbox. This window contains all design time properties, including the extended properties that Architect adds.

The properties available in the Toolbox window are described in the table below:

Property Name	Type	Definition
DesignMode	logical	Determines whether the PAReports module shows the report design interface.
MessageTitle	string	Sets the title that will appear on any messages displayed by the PAReports object. The default is DataPA.
Name	string	The Name property contains the name of the control. The name is important because it identifies the control. You can use the control's name to access the properties and methods of the control.
OutlookbarVisible	Logical	Determines whether the PAReports outlookbar is visible. Default is TRUE.
PreviewMode	logical	Determines whether the PAReports module shows the report preview interface.
PreviewZoom	Integer	The PreviewZoom property controls the extent to which the report preview window zooms into the image of the report.

Property Name	Type	Definition
RulersVisible	Logical	The RulersVisible property controls whether or not the rulers are visible in the preview pane of the PAReports control.
StatusBarVisible	Logical	The StatusBarVisible property controls whether or not the status bar is visible at the base of the PAReports control.
Parent	Handle	The Parent property is the handle of the container in which the control resides. This property is set internally by Visual Studio.
Tag	string	The Tag property is a user property that allows the user to store an arbitrary string value and retrieve it later. Progress does not use this property internally, and it is intended to give the user a way of storing application specific information with the control. This property is initialized to an empty string.
ToolBarsVisible	logical	The ToolBarsVisible property controls whether or not the tool bars are visible at the top of the PAReports control.
Visible	Logical	The Visible property determines and indicates whether a PAReports control is currently displayed. The Visible property will appear in the Property Toolbox and can be set at design time. It defaults to TRUE. The value set in the Property Editor determines whether the control is initially displayed when the project is run.



Referencing Properties & Methods at Runtime

The syntax to access properties and methods at runtime is similar to the syntax to access widget attributes and methods. However, where widget references use widget handles, PAReports object references use component handles. Component handles support an extended syntax that allows you to chain component handle references to properties and methods.

AppBuilder Property Syntax

The syntax for referencing a property of the PAReports COM object is:

```
COMhdl-expression:Property-Name-Reference
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

Property-Name-Reference
Specifies a single PAReports property.

The following example assigns a value of 75 to the PreviewZoom property of our PAReports object:

```
PAReports:PreviewZoom = 75.
```

Visual Studio Property Syntax

The syntax for referencing this property of the PAReports COM object is:

```
Control.Property-Name-Reference
```

Control
Is an expression that returns a component handle of a realised PAReports instance.

Property-Name-Reference
Specifies a single PAReports property.

The following example assigns a value of 75 to the PreviewZoom property of our PAReports object:

```
PAReports.PreviewZoom = 75
```

Architect Property Syntax

The syntax for referencing a property of the PAReports COM object is:

```
Control:Property-Name-Reference
```

Control
Is an expression that returns a component handle of a realised PAReports instance.

Property-Name-Reference
Specifies a single PAReports property.

The following example assigns a value of 75 to the PreviewZoom property of our PAReports object:

```
PAReports:PreviewZoom = 75.
```

As mentioned above, this syntax can be extended to string together several properties to reference a single property. The following example uses the ApplicationObject property of the PAReports object, and the Query property of the ApplicationObject to return the com-handle of the query used in the current report:

AppBuilder Example

```
chCurrentQuery = PAReports:ApplicationObject:Query
```

Visual Studio Example

```
Query = PAReports.ApplicationObject.Query
```

Architect Example

```
Query = PAReports:ApplicationObject:Query()
```

AppBuilder Method Syntax

The syntax for referencing a method of the PAReports COM object is:

```
COMhdl-expression:Method-Name
(
        {       [OUTPUT | INPUT-OUTPUT ]
                        expression | null-parameter
        }
        [ ,     [OUTPUT | INPUT-OUTPUT ]
                        expression | null-parameter
        ] . . .
)
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

Method-Name
The name of the COM object method.

expression
Any valid Progress expression that you can pass as a parameter to the method.

null-parameter
Any amount of white space, indicating an optional parameter that you choose to omit.

The following example calls the OpenReport method to open the report specified by the string in the cReportFile variable, and passing in a value of TRUE to indicate that we do want the report designer to prompt the user to save any changes to an existing report if it is open:

```
PAReports:OpenReport(cReportFile, TRUE).
```

Visual Studio Method Syntax

The syntax for referencing a method of the PAReports COM object is:

```
Control:Method-Name
(
        [ ,     expression | null-parameter
        ] . . .
)
```

Control
Is an expression that returns a component handle of a realised PAReports instance.

Method-Name
The name of the .NET control method.

expression
Any valid expression that you can pass as a parameter to the method.

null-parameter
Any amount of white space, indicating an optional parameter that you choose to omit.

The following example calls the OpenReport method to open the report specified by the string in the cReportFile variable, and passing in a value of TRUE to indicate that we do want the report designer to prompt the user to save any changes to an existing report if it is open:

```
PAReports.OpenReport(sReportFile, TRUE)
```

Architect Method Syntax

The syntax for referencing a method of the PAReports control in Architect object is:

```
Control:Method-Name
(
        {       [OUTPUT | INPUT-OUTPUT ]
                    expression | null-parameter
        }
        [ ,     [OUTPUT | INPUT-OUTPUT ]
                    expression | null-parameter
        ] . . .
)
```

Control
Is an expression that returns a component handle of a realised PAReports instance.

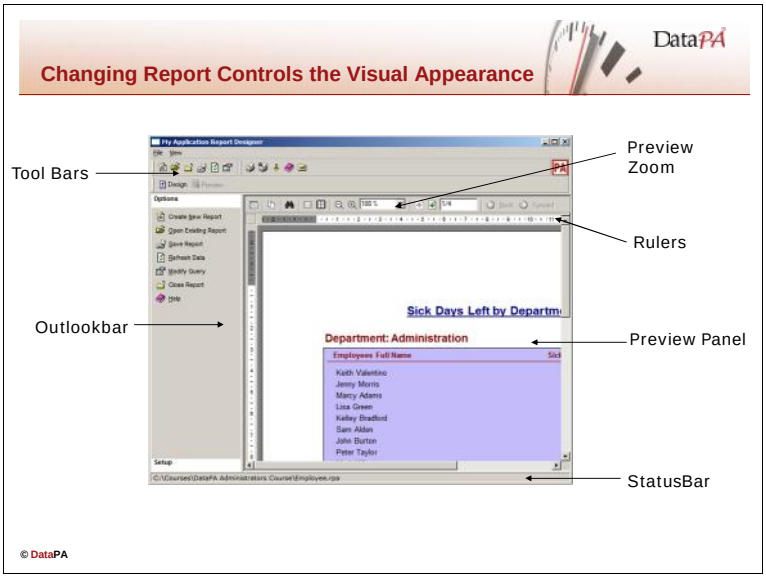
Method-Name
The name of the .NET control method.

expression
Any valid Progress expression that you can pass as a parameter to the method.

null-parameter
Any amount of white space, indicating an optional parameter that you choose to omit.

The following example calls the OpenReport method to open the report specified by the string in the cReportFile variable, and passing in a value of TRUE to indicate that we do want the report designer to prompt the user to save any changes to an existing report if it is open:

```
PAReports:OpenReport(cReportFile, TRUE).
```



Changing Report Controls Visual Appearance

The PAReports object provides five properties to change the appearance of the object in the user interface. These properties can be used to configure the PAReports object for particular uses (e.g. as a report viewer or a report designer), or exposed to the user through other user-interface objects to allow the user to configure the object to suit their requirements.

The OutlookbarVisible property

The OutlookbarVisible property allows you to show or hide the outlook bar on the left hand side of the PAReports object.

AppBuilder Property Syntax

The syntax for referencing this property of the PAReports COM object is:

```
COMhdl-expression:OutlookbarVisible = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

expression
Any valid Progress logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the outlookbar when the appropriate menu-item is checked or unchecked:

```
PAReports:OutlookBarVisible = MENU-ITEM m_Show_OutlookBar:CHECKED
                                        IN MENU MENU-BAR-C-Win.
```

Visual Studio Property Syntax

The syntax for referencing this property of the PAReports .NET control from Visual Studio is:

```
Control.OutlookbarVisible = expression
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid .NET logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the outlookbar when the appropriate menu-item is checked or unchecked:

```
PAReports:OutlookBarVisible = mnuShowOutlookbar.checked
```

Architect Property Syntax

The syntax for referencing this property of the PAReports .NET control from Architect is:

```
Control:OutlookbarVisible = expression.
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid Progress logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the outlookbar when the appropriate menu-item is checked or unchecked:

```
PAReports:OutlookBarVisible = mnuShowOutlookbar:checked.
```

The RulersVisible property

The RulersVisible property allows you to show or hide the rulers in the preview window. (NB. The RulersVisible property can only be set to true if the PreviewZoom is set high enough to give measurable values) .

AppBuilder Property Syntax

The syntax for referencing this property of the PAReports COM object is:

```
COMhdl-expression:RulersVisible = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

expression
Any valid Progress logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the preview panel rulers when the appropriate menu-item is checked or unchecked:

```
PAReports:RulersVisible = MENU-ITEM m_Show_Rulers:CHECKED IN MENU
                                    MENU-BAR-C-Win.
```

Visual Studio Property Syntax

The syntax for referencing this property of the PAReports .NET control from Visual Studio is:

```
Control.RulersVisible = expression
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid .NET logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the rulers when the appropriate menu-item is checked or unchecked:

```
PAReports.RulersVisible = mnuShowRulers.checked
```

Architect Property Syntax

The syntax for referencing this property of the PAReports .NET control from Architect is:

```
Control:RulersVisible = expression.
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid Progress logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the rulers when the appropriate menu-item is checked or unchecked:

```
PAReports:RulersVisible = mnuShowRulers:checked.
```

The StatusbarVisible property

The StatusbarVisible property allows you to show or hide the status bar at the bottom of the PAReports object.

AppBuilder Property Syntax

The syntax for referencing this property of the PAReports COM object is:

```
COMhdl-expression:StatusbarVisible = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

expression
Any valid Progress logical expression that resolves to TRUE or FALSE.

For example, the following code hides the status bar:

```
PAReports:StatusbarVisible = FALSE.
```

Visual Studio Property Syntax

The syntax for referencing this property of the PAReports .NET control from Visual Studio is:

```
Control.StatusbarVisible = expression
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid .NET logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the status bar when the appropriate menu-item is checked or unchecked:

```
PAReports.StatusbarVisible = mnuShowStatusBar.checked
```

Architect Property Syntax

The syntax for referencing this property of the PAReports .NET control from Architect is:

```
Control:StatusbarVisible = expression.
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid Progress logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the rulers when the appropriate menu-item is checked or unchecked:

```
PAReports:StatusbarVisible = mnuShowStatusbar:checked.
```

The ToolbarsVisible property

The ToolbarsVisible property allows you to show or hide the toolbars at the top of the PAReports object.

AppBuilder Property Syntax

The syntax for referencing this property of the PAReports COM object is:

```
COMhdl-expression:ToolbarsVisible = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

expression
Any valid Progress logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the toolbars when the appropriate menu-item is checked or unchecked:

```
PAReports:ToolbarsVisible = MENU-ITEM m_Show_Toolbars:CHECKED
                                    IN MENU MENU-BAR-C-Win.
```

Visual Studio Property Syntax

The syntax for referencing this property of the PAReports .NET control from Visual Studio is:

```
Control.ToolbarsVisible = expression
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid .NET logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the tool bars when the appropriate menu-item is checked or unchecked:

```
PAReports.ToolbarsVisible = mnuShowToolbars.checked
```

Architect Property Syntax

The syntax for referencing this property of the PAReports .NET control from Architect is:

```
Control:ToolbarsVisible = expression.
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid Progress logical expression that resolves to TRUE or FALSE.

For example, the following code shows or hides the tool bars when the appropriate menu-item is checked or unchecked:

```
PAReports:ToolbarsVisible = mnuShowToolbars:checked.
```

The PreviewZoom property

The PreviewZoom property allows you to level of zoom used to view the report in the report preview panel.

AppBuilder Property Syntax

The syntax for referencing this property of the PAReports COM object is:

```
COMhdl-expression:PreviewZoom = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

expression
Any valid Progress integer expression.

The following example assigns a value of 75 to the PreviewZoom property of our PAReports object:

```
PAReports:PreviewZoom = 75.
```

Visual Studio Property Syntax

The syntax for referencing this property of the PAReports .NET control from Visual Studio is:

```
Control.PreviewZoom = expression
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid .NET integer expression.

The following example assigns a value of 75 to the PreviewZoom property of our PAReports object:

```
PAReports.PreviewZoom = 75
```

Architect Property Syntax

The syntax for referencing this property of the PAReports .NET control from Architect is:

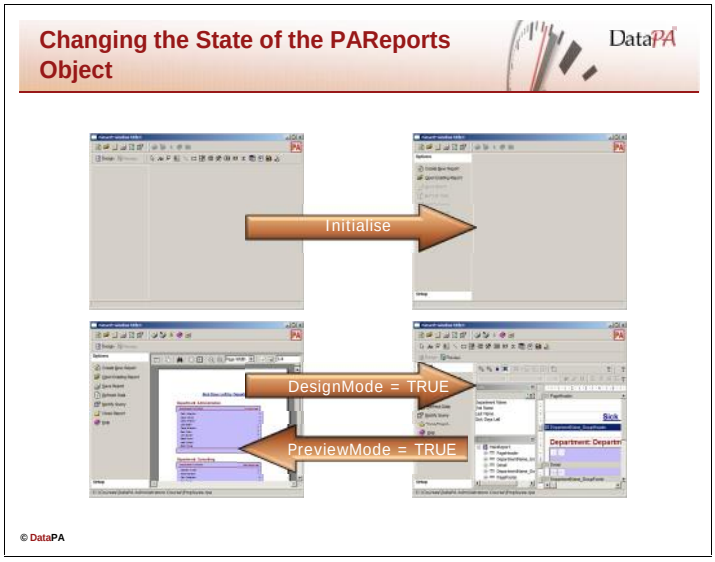
```
Control:PreviewZoom = expression.
```

Control
Is an expression that returns handle of a realised PAReports instance.

expression
Any valid Progress integer expression.

The following example assigns a value of 75 to the PreviewZoom property of our PAReports object:

```
PAReports:PreviewZoom = 75.
```



Changing the State of the PAReports Object

The PAReports object provides a method and two properties to change the state of the PAReports object.

The Initialise method

The Initialise method must be called before the PAReports object can be used.

AppBuilder Method Syntax

It is best practice to call the initialise method in the initialise-controls procedure in your Progress GUI window. The syntax for referencing this property of the PAReports COM object is:

```
COMhdl-expression:Initialise()
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

The following example shows the use of the initialise method in our example application:

```
PAReports:Initialise().
```

Visual Studio Method Syntax

It is best practice to call the initialise method in the load event of your .NET form. The syntax for referencing this property of the PAReports object is:

```
Control.Initialise()
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the initialise method in our example application:

```
PAReports.Initialise()
```

Architect Method Syntax

It is best practice to call the initialise method in the load event of your ABL .NET form. The syntax for referencing this property of the PAReports object is:

```
Control:Initialise().
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the initialise method in our example application:

```
PAReports:Initialise().
```

The DesignMode Property

The DesignMode property allows you to set the PAReports control into design mode or query whether or not the PAReports control is in design mode. The DesignMode property should only be set when there is a report open in the control.

AppBuilder Property Syntax

The syntax for referencing this property of the PAReports COM object is:

```
COMhdl-expression:DesignMode = TRUE
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

The following example shows the use of the DesignMode property in our example application:

```
PAReports:DesignMode = TRUE.
```

Visual Studio Property Syntax

The syntax for referencing this property of the PAReports object is:

```
Control.DesignMode = TRUE
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the DesignMode property in our example application:

```
PAReports.DesignMode = TRUE
```

Architect Property Syntax

The syntax for referencing this property of the PAReports object is:

```
Control:DesignMode = TRUE
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the DesignMode property in our example application:

```
PAReports:DesignMode = TRUE.
```

The PreviewMode Property

The PreviewMode property allows you to set the PAReports control into preview mode or query whether or not the PAReports control is in preview mode.

AppBuilder Property Syntax

The syntax for referencing this property of the PAReports COM object is:

```
COMhdl-expression:PreviewMode = TRUE
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

The following example shows the use of the PreviewMode property in our example application:

```
PAReports:PreviewMode = TRUE.
```

Visual Studio Property Syntax

The syntax for referencing this property of the PAReports object is:

```
Control.PreviewMode = TRUE
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the PreViewMode property in our example application:

```
PAReports.PreviewMode = TRUE
```

Architect Property Syntax

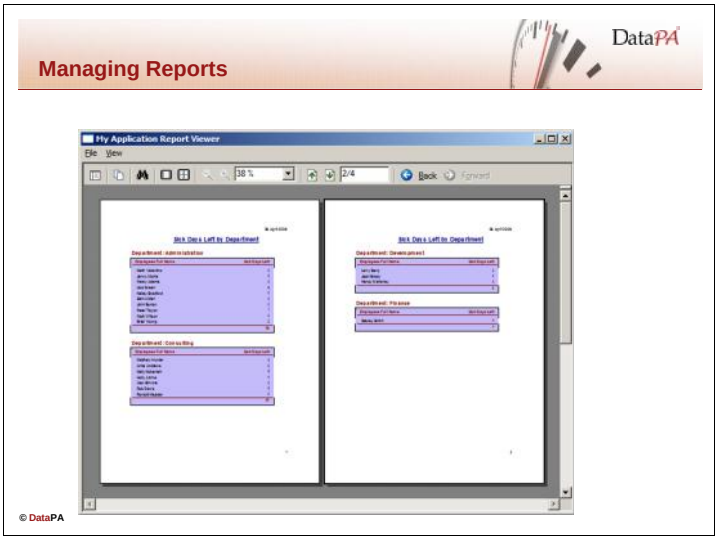
The syntax for referencing this property of the PAReports object is:

```
Control:PreviewMode = TRUE
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the PreviewMode property in our example application:

```
PAReports:PreviewMode = TRUE.
```



Managing Reports

The PAReports control provides six methods to allow the developer to manage reports with the PAReports control. The methods can be used to automatically manage reports using the application code, or respond to user interface events to provide functionality for the user.

The OpenReport Method

The OpenReport method opens a report in the PAReports control.

AppBuilder Method Syntax

The syntax for referencing this method of the PAReports COM object is:

```
COMhdl-expression:OpenReport(ReportName, CheckBeforeClose)
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

ReportName
A string expression that returns the fully qualified pathname to the required report.

CheckBeforeClose
A logical expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the OpenReports method in our example application:

```
    PAReports:OpenReport(cReportFile, TRUE).
```

Visual Studio Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.OpenReport(ReportName, CheckBeforeClose)
```

Control
Is an expression that returns handle of a realised PAReports instance.

ReportName
A string expression that returns the fully qualified pathname to the required report.

CheckBeforeClose
A boolean expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the OpenReports method in our example application:

```
PAReports.OpenReport(cReportFile, TRUE)
```

Architect Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control:OpenReport(ReportName, CheckBeforeClose)
```

Control
Is an expression that returns handle of a realised PAReports instance.

ReportName
A string expression that returns the fully qualified pathname to the required report.

CheckBeforeClose
A logical expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the OpenReports method in our example application:

```
PAReports:OpenReport(cReportFile, TRUE).
```

The CloseReport Method

The CloseReport method closes a report in the PAReports control.

AppBuilder Method Syntax

The syntax for referencing this method of the PAReports COM object is:

```
COMhdl-expression:CloseReport(CheckBeforeClose)
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

CheckBeforeClose
A logical expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the CloseReports method in our example application:

```
PAReports:CloseReport(TRUE).
```

Visual Studio Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.CloseReport(CheckBeforeClose)
```

Control
Is an expression that returns handle of a realised PAReports instance.

CheckBeforeClose
A boolean expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the CloseReports method in our example application:

```
PAReports.CloseReport(TRUE)
```

Architect Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control:CloseReport(CheckBeforeClose)
```

Control
Is an expression that returns handle of a realised PAReports instance.

CheckBeforeClose
A logical expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the CloseReports method in our example application:

```
PAReports:CloseReport(TRUE).
```

The NewReport Method

The NewReport method opens a new report in the PAReports control.

AppBuilder Method Syntax

The syntax for referencing this methos of the PAReports COM object is:

```
COMhdl-expression:NewReport(CheckBeforeClose)
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

CheckBeforeClose
A logical expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the NewReport method in our example application:

```
PAReports:NewReport(TRUE).
```

Visual Studio Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.NewReport(CheckBeforeClose)
```

Control
Is an expression that returns handle of a realised PAReports instance.

CheckBeforeClose
A boolean expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the NewReport method in our example application:

```
PAReports.NewReport(TRUE)
```

Architect Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control:NewReport(CheckBeforeClose)
```

Control
Is an expression that returns handle of a realised PAReports instance.

CheckBeforeClose
A logical expression. If the expression is TRUE, PAReports will check if there is an existing report open with any changes and prompt the user to save the changes before continuing.

The following example shows the use of the NewReport method in our example application:

```
PAReports:NewReport(TRUE).
```

The SaveReport Method

The SaveReport method saves the report in the PAReports control.

AppBuilder Method Syntax

The syntax for referencing this method of the PAReports COM object is:

```
COMhdl-expression:SaveReport(SaveAs)
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

SaveAs
A logical expression. If the expression is TRUE, PAReports show the Save As dialog box, otherwise, PAReports will only show the Save As dialog box if the report has not been previously saved.

The following example shows the use of the SaveReport method in our example application:

```
PAReports:SaveReport(FALSE).
```

Visual Studio Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.SaveReport(SaveAs)
```

Control
Is an expression that returns handle of a realised PAReports instance.

SaveAs
A boolean expression. If the expression is TRUE, PAReports show the Save As dialog box, otherwise, PAReports will only show the Save As dialog box if the report has not been previously saved.

The following example shows the use of the SaveReport method in our example application:

```
PAReports.SaveReport(FALSE)
```

Architect Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control:SaveReport(SaveAs)
```

Control
Is an expression that returns handle of a realised PAReports instance.

SaveAs
A logical expression. If the expression is TRUE, PAReports show the Save As dialog box, otherwise, PAReports will only show the Save As dialog box if the report has not been previously saved.

The following example shows the use of the SaveReport method in our example application:

```
PAReports:SaveReport(FALSE).
```

The RefreshData Method

The RefreshData method refreshes the data in the current report.

AppBuilder Method Syntax

The syntax for referencing this method of the PAReports COM object is:

```
COMhdl-expression:RefreshData()
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

The following example shows the use of the Refreshdata method in our example application:

```
PAReports:RefreshData().
```

Visual Studio Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.RefreshData()
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the Refreshdata method in our example application:

```
PAReports.RefreshData()
```

Architect Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control:RefreshData()
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the Refreshdata method in our example application:

```
PAReports:RefreshData().
```

The PrintReport Method

The PrintReport method prints the report in the PAReports control.

AppBuilder Method Syntax

The syntax for referencing this method of the PAReports COM object is:

```
COMhdl-expression:PrintReport(DisplayDialog)
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

DisplayDialog
A logical expression. If the expression is TRUE, PAReports will display the print report dialog box to allow the user to select the print options. Otherwise the PAReports option use the default print settings to print the report.

The following example shows the use of the PrintReport method in our example application:

```
PAReports:PrintReport(TRUE).
```

Visual Studio Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.PrintReport(DisplayDialog)
```

Control
Is an expression that returns handle of a realised PAReports instance.

DisplayDialog
A boolean expression. If the expression is TRUE, PAReports will display the print report dialog box to allow the user to select the print options. Otherwise the PAReports option use the default print settings to print the report.

The following example shows the use of the PrintReport method in our example application:

```
PAReports.PrintReport(TRUE)
```

Architect Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.PrintReport(DisplayDialog)
```

Control
Is an expression that returns handle of a realised PAReports instance.

DisplayDialog
A logical expression. If the expression is TRUE, PAReports will display the print report dialog box to allow the user to select the print options. Otherwise the PAReports option use the default print settings to print the report.

The following example shows the use of the PrintReport method in our example application:

```
PAReports:PrintReport(TRUE).
```

The PrintSetup Method

The PrintSetup method shows the print setup dialog box in the PAReports control.

AppBuilder Method Syntax

The syntax for referencing this method of the PAReports COM object is:

```
COMhdl-expression:PrintSetup()
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

The following example shows the use of the PrintSetup method in our example application:

```
PAReports:PrintSetup().
```

Visual Studio Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.PrintSetup()
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the PrintSetup method in our example application:

```
PAReports.PrintSetup()
```

Architect Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control:PrintSetup()
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the PrintSetup method in our example application:

```
PAReports:PrintSetup().
```

The ExportReport Method

The ExportReport method shows the export dialog box from the PAReports control. This will allow the user to export the report to a range of different formats.

AppBuilder Method Syntax

The syntax for referencing this method of the PAReports COM object is:

```
COMhdl-expression:ExportReport()
```

COMhdl-expression
Is an expression that returns a component handle of a realised PAReports instance.

The following example shows the use of the Export method in our example application:

```
PAReports:ExportReport().
```

Visual Studio Method Syntax

The syntax for referencing this method of the PAReports object is:

```
Control.ExportReport()
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the Export method in our example application:

```
PAReports.ExportReport()
```

Architect Method Syntax

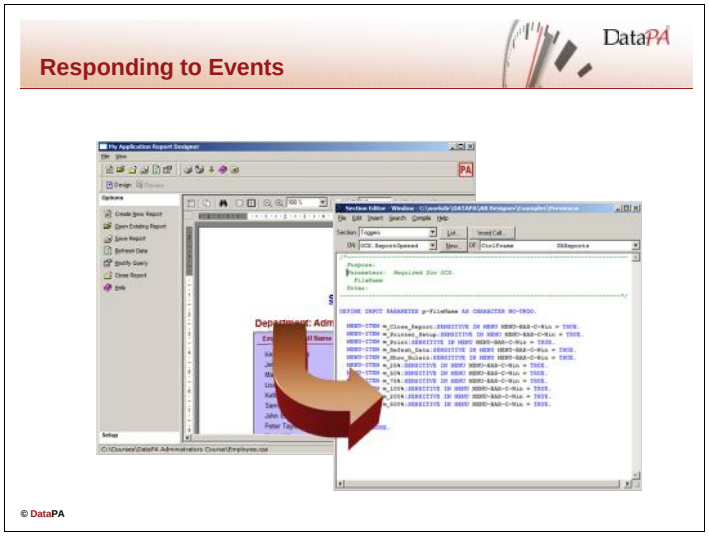
The syntax for referencing this method of the PAReports object is:

```
Control:ExportReport()
```

Control
Is an expression that returns handle of a realised PAReports instance.

The following example shows the use of the Export method in our example application:

```
PAReports:ExportReport().
```



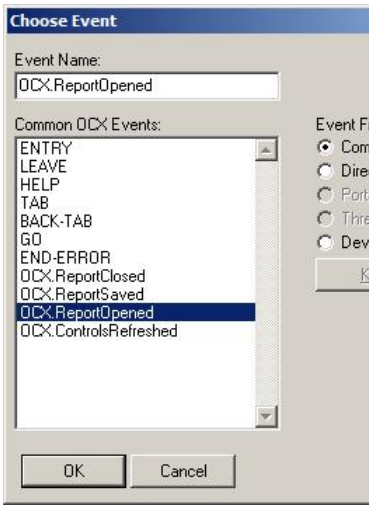
Responding to Events in the AppBuilder

Progress handles PAReports events using OCX event procedures. An *OCX event procedure* is a standard Progress internal procedure that serves as an event handler for ActiveX controls. PAReports will often pass one or more parameters that must be handled by the event procedure. These parameters are defined as parameters in the event procedure. Progress identifies an OCX event procedure from the construction of its name. This is the only syntactic feature that distinguishes Progress internal procedures as an OCX event procedure.

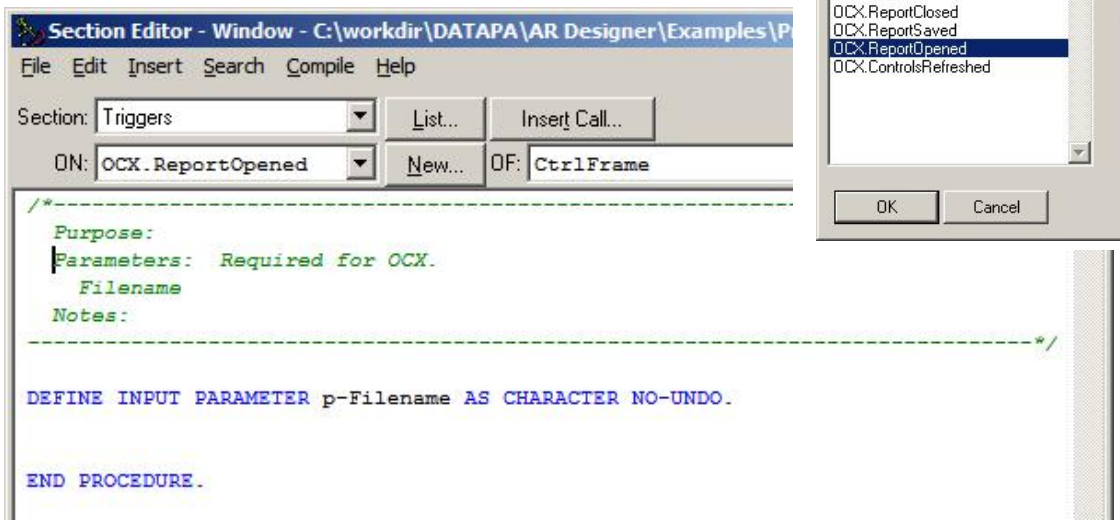
Creating Event Procedures in the AppBuilder

Follow these steps to create an OCX event procedure in the AppBuilder:

- Click on the edit code button in the AppBuilder to open the *section editor*
- Select *Triggers* in the *Section* combo-box
- Select the control frame in the *OF* combo-box
- Press New
- Select the appropriate event



The AppBuilder will create an appropriate event procedure with the relevant paramaters…



Responding to Events in Visual Studio

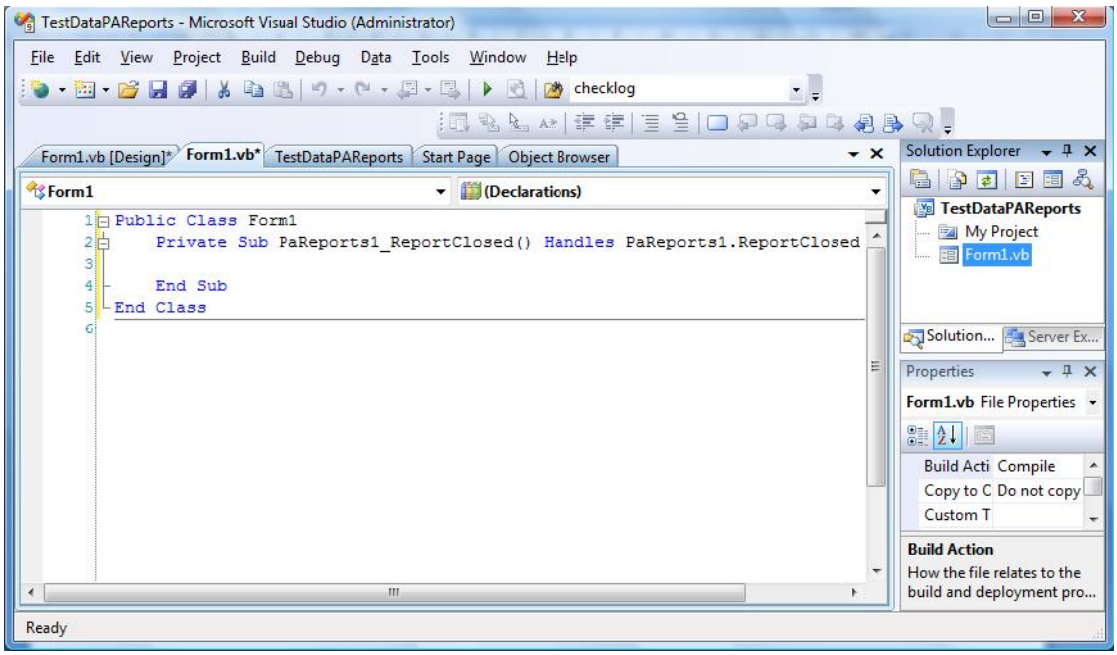
Visual Studio handles PAReports events using standard .NET event procedures. PAReports will often pass one or more parameters that must be handled by the event procedure. These parameters are defined as parameters in the event procedure. Visual Studio identifies an event procedure from handler that is assigned to it for example (`Handles PaReports1.ReportClosed`).

Creating Event Procedures in Visual Studio

Follow these steps to create an event procedure in Visual Studio:

- Select the View Code option for the form which hosts the PAReports control
- Select the PAReports instance from the object drop down
- Select the event from the drop down list of events beside it

Visual Studio will create an appropriate event procedure with the correct handler assigned.



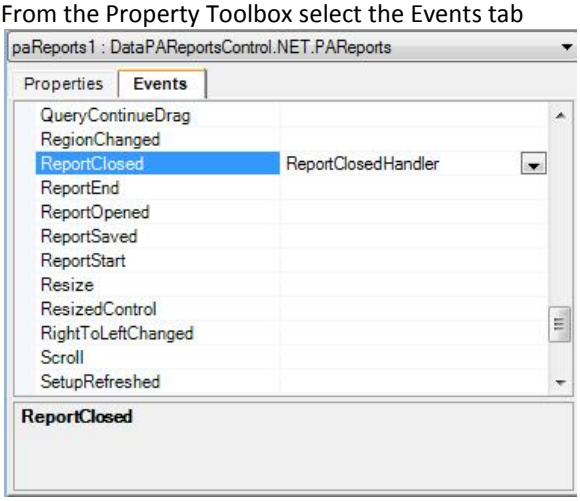
Responding to Events in Architect

Architect handles PAReports events using Architect event procedures. PAReports will often pass one or more parameters that must be handled by the event procedure. These parameters are defined as parameters in the event procedure. Architect identifies an event procedure using the subscribe statement that is set in the InitializeComponent method of the ABL Form. For example (`THIS-OBJECT:paReports1:ReportClosed:SUBSCRIBE(THIS-OBJECT:ReportClosedHandler).`). The code to handle the event should then be placed in the method created by Architect.

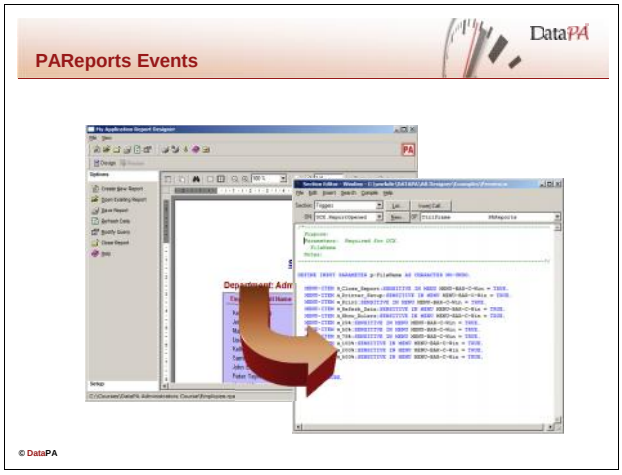
Creating Event Procedures in Architect

Follow these steps to create an event procedure in Architect:

- Select the instance of PAReports in the ABL Form designer
- From the Property Toolbox select the Events tab



- Against the event you want to add event procedure for enter the name of the procedure you want to use.
- Architect will add a new method to the ABL Form which will have the name entered.



Introduction

PAReports provides nine events to allow the developer to respond to changes in the state of the object.

PAReports Events

These events are listed in the table below:

Event Name	Parameters	Definition & Usage
ControlsRefreshed	None	Event is triggered when any code or user interaction causes the controls in the PAReports object to be redrawn. The event is useful to respond to changes in state of the object. In our example application the ControlsRefreshed event is used to check and uncheck m_Show_Rulers, m_Show_OutlookBar and m_Show_Toolbars menu items to reflect whether or not these items are visible in the PAReports object.
ReportOpened	Filename (String)	Event is triggered when a report is opened, with the input parameter receiving the name of the report that has just been opened. In our example the ReportOpened event is used to enable widgets that are only relevant if a report is open in the PAReports object.
ReportClosed	None	Event is triggered when a report is closed. In our example the ReportClosed event is used to disable widgets that are only relevant if a report is open in the PAReports object.
ReportSaved	Filename (String)	Event is triggered when a report is saved, with the input parameter receiving the name of the report that has just been opened. The event is useful for situations where you want to record reports that have been saved, for example when building a list most recently used reports.
ReportStart	None	Event is triggered when a report refresh is started.

Event Name	Parameters	Definition & Usage
ReportEnd	None	Event is triggered when a report refresh is ended. This event is useful for knowing when a report refresh has been completed so that another action on the report is not started while the report is still updating.
LoadCompleted	None	This event is triggered when the load of the report viewer is completed and all the pages of the report and loaded into the report viewer.
DesignerStatusChange	Action (integer)	This event is triggered each time the status of the report designer changes. The parameter Action is passed in and then denotes the type of change that has been made. This is useful so that when a change is made an action can be programmed in response to the status change.
SetupRefreshed	None	This event is triggered when the DataPA setup files are refreshed. This is useful for knowing when a potential problem

The settings for *action* are:

Value	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

DataPA Enterprise Dashboard Control

Introduction

The DataPA Enterprise Dashboard control allows the DataPA dashboard designer and viewer window to be embedded into any application to provide dashboard functionality. This chapter concentrates on using the DataPA Enterprise Dashboard control with Progress GUI, Visual Studio and OpenEdge Architect clients.

Learning Objectives

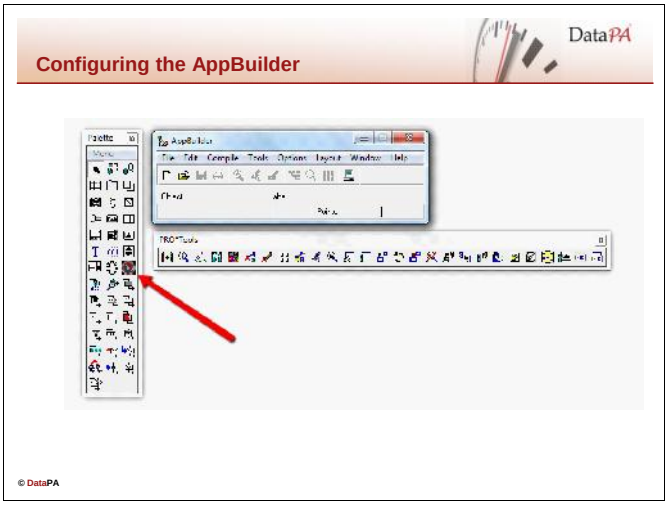
When you complete this lesson you should be able to:

- Configure your development environment to use the DataPA Enterprise Dashboard control.
- Add the DataPA Enterprise Dashboard control to a form.
- Use the controls properties and methods to control its behaviour.
- Respond to events triggered by the DataPA Enterprise Dashboard controls.

Prerequisites

Before you begin this lesson you should be able to:

- Create a GUI application using Progress
- Setup and configure DataPA for an AppServer application

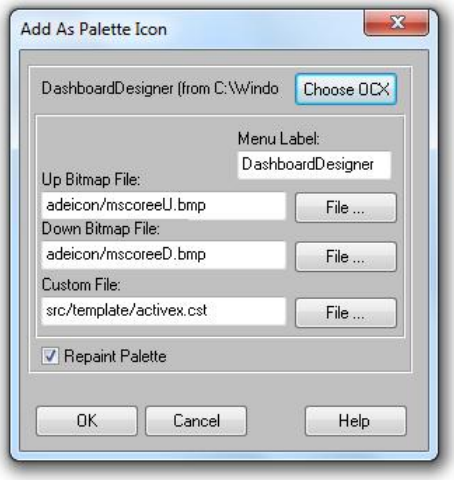


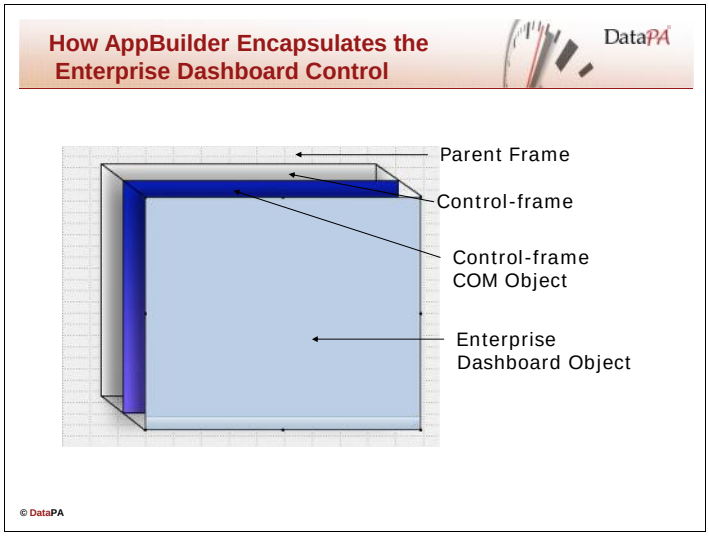
Configuring Progress AppBuilder

To provide quick and simple access to the DataPA Enterprise Dashboard Control in the AppBuilder, we can add it to the AppBuilder Object Palette.

Follow these steps to add the DataPA Enterprise Dashboard Control to the AppBuilder.

- Start a copy of the the Progress AppBuilder
- From the AppBuilder Object Palette menu, select Add OCX
- Add as Palette Icon
- Press the Choose OCX button
- Select DataPAEnterpriseDashboard.DashboardDesigner from the selection list
- Press OK to close the list of available OCX objects
- Press OK to close the Add As Palette Icon dialog box.





How the AppBuilder encapsulates the DataPA Enterprise Dashboard Control

The DataPA Enterprise Dashboard Control is an ActiveX control or OCX. To comply with the COM object standard, an ActiveX control instance must be placed in a *control container* that handles events and specific user-interface functionality for the control. Progress supports this standard with the control-frame widget, the container, called a control-frame COM object and the control itself, in our case a DashboardDesigner object.

Control-frame Widget

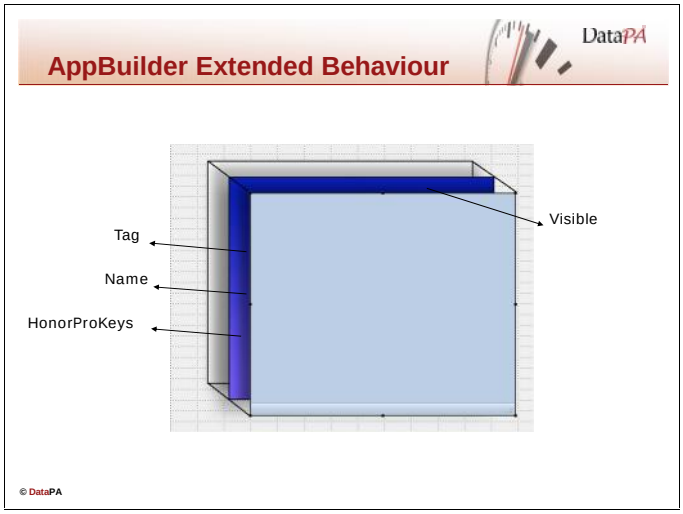
The control frame widget is a specialised Progress frame that establishes the relationship between a DashboardDesigner control, the other Progress widgets in the user interface, and the 4GL. As such the control-frame gives access to properties that interact with the other 4GL widgets, (like ROW and TAB-POSITION), and events that interact with other widgets (like TAB and LEAVE) or have specific relevance in the 4GL (like GO and END-ERROR).

The control-frame has the following attributes:

Widget Attribute	COM Object Property
HEIGHT-PIXELS, HEIGHT[-CHARS]	Height
NAME	Name
WIDTH-PIXELS, WIDTH[-CHARS]	Width
X, COLUMN	Left
Y, ROW	Top

Control-frame COM Object

The control-frame COM object is the actual DashboardDesigner control container. It provides the initial point of access to the DashboardDesigner control from the 4GL. Through the COM object you can access the component handle of the DashboardDesigner control to directly access its properties and methods.



AppBuilder Extended Behaviour

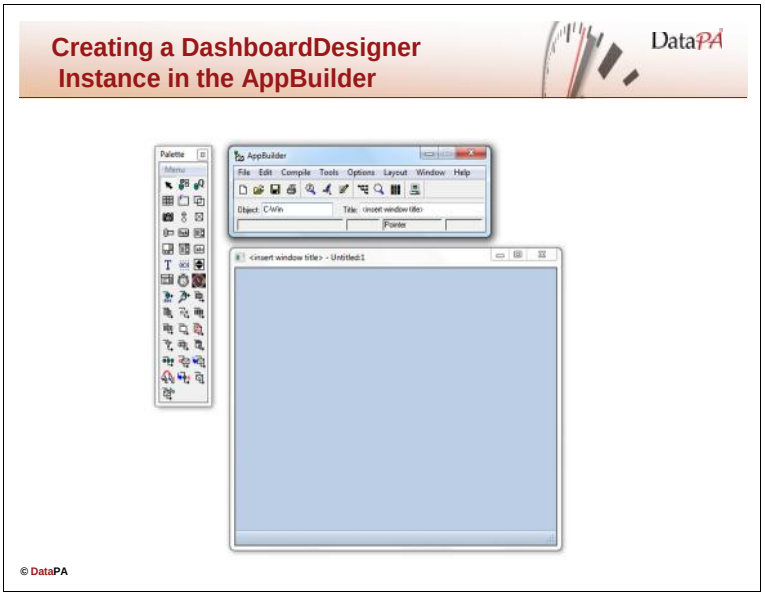
In addition to providing an initial point of access for the DashboardDesigner control, the control-frame COM object acts as a wrapper to the DashboardDesigner com object providing extended functionality specific to the Progress environment. Even though the control-frame COM object provides this extended functionality, and the functionality is completely unknown to the DashboardDesigner com object, the extended methods, properties and events appear as though part of the DashboardDesigner object itself and are accessed through the same com object handle (com object handles will be discussed later in this lesson).

Extended Properties

The table below lists the extended properties available for the DashboardDesigner control:

Property Name	Type	Definition
HonorProKeys	logical	Default is TRUE. Determines who processes the GO, ENDKEY, HELP, and TAB keys: Progress, or the DashboardDesigner control. If the property is TRUE, Progress intercepts these keys and processes them as normal Progress key events. If the property is FALSE, the keystrokes are sent to the DashboardDesigner control for processing.
HonorReturnKey	logical	Default is FALSE. If the property is TRUE, Progress intercepts the key and processes it as a normal Progress RETURN key event. If the property is FALSE, which is the default, the keystroke is sent to the DashboardDesigner control for processing.
Name	string	The Name property contains the name of the control. The name is important because it identifies the control. You can use the control's name to get a COM-HANDLE to the control (for example, ASSIGN chDashboard = chCtrlFrame: DashboardDesigner, where DashboardDesigner is the control's name and chCtrlFrame is the control frame handle). The control name associates event handlers with a control.
Parent	COM-Handle	The Parent property is the com-handle of the container in which the control resides. This property is set internally by Progress.

Property Name	Type	Definition
Tag	string	The Tag property is a user property that allows the user to store an arbitrary string value and retrieve it later. Progress does not use this property internally, and it is intended to give the user a way of storing application specific information with the control. This property is initialized to an empty string.
Visible	Logical	The Visible property determines and indicates whether a DashboardDesigner control is currently displayed. The Visible property is distinct from, but influenced by, the Visible and Hidden attributes of the Control-Frame widget. The Visible property will appear in the Property Editor and can be set at design time. It defaults to TRUE. The value set in the Property Editor determines whether the OCX is initially displayed when the program is run, but can be overridden by the value of the Control-Frame widget's Hidden attribute.



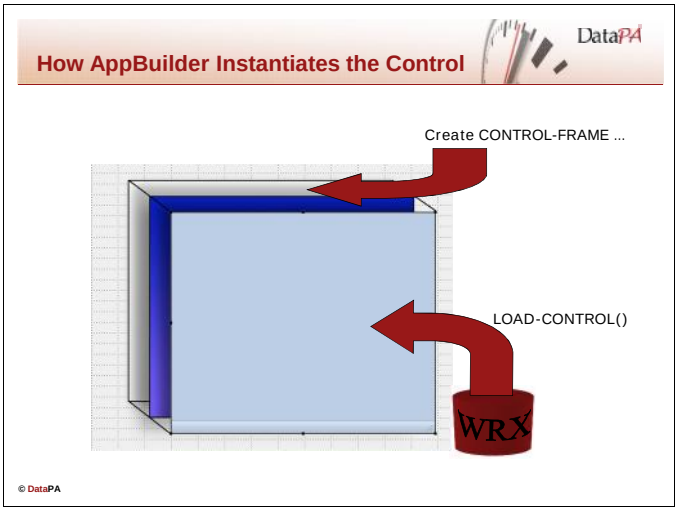
Creating a DashboardDesigner Instance in the AppBuilder

You must use the AppBuilder to add a DashboardDesigner control to a Progress user interface. Follow these steps in the AppBuilder to add a DashboardDesigner control:

- Open a container object, such as a Window, Dialog or SmartWindow.
- Choose the DashboardDesigner ActiveX control from the object palette.
- Click on the container object to place the DashboardDesigner control onto the container.
- Resize the DashboardDesigner control as required.

If you save your work at this point, the AppBuilder generates the following data and code for your application:

- The definition for the default instance of the DashboardDesigner control in a binary (.wrx) file. This includes the initial definition provided by DataPA.
- The default 4GL code in your .w file to instantiate and orient the DashboardDesigner control in the user interface at runtime.



How the AppBuilder Instantiates the Control

At runtime, Progress uses AppBuilder-generated code to instantiate the DashboardDesigner control. This code uses the CREATE Widget statement to realize a control-frame widget, initializing the control-frame with the name specified at design time. It then invokes the LoadControls( ) method on the control-frame COM object to instantiate the PAReports control, loading all design time definitions from the .wrx file.

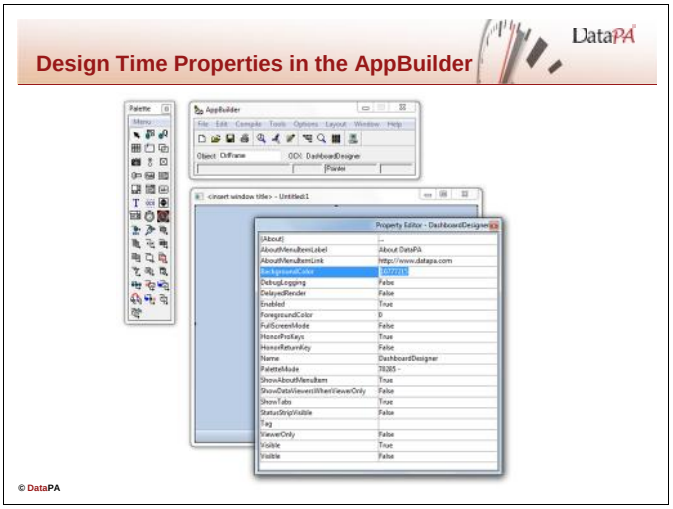
Example

The following AppBuilder-generated code creates the control-frame in our example application:

```
CREATE CONTROL-FRAME CtrlFrame ASSIGN
       FRAME           = FRAME DEFAULT-FRAME:HANDLE
       ROW             = 1
       COLUMN          = 1
       HEIGHT          = 21.67
       WIDTH           = 137
       WIDGET-ID       = 2
       HIDDEN          = no
       SENSITIVE       = yes.
      CtrlFrame:NAME = "CtrlFrame":U .
```

The following code calls the LoadControls method to load the control into the control-frame COM object.

```
DEFINE VARIABLE UIB_S    AS LOGICAL    NO-UNDO.
DEFINE VARIABLE OCXFile  AS CHARACTER  NO-UNDO.

OCXFile = SEARCH( "{&FILE-NAME}.wrx":U ).
IF OCXFile = ? THEN
  OCXFile = SEARCH(SUBSTRING(THIS-PROCEDURE:FILE-NAME, 1,
                    R-INDEX(THIS-PROCEDURE:FILE-NAME, ".":U), "CHARACTER":U)
+ "wrx":U).

IF OCXFile <> ? THEN
DO:
  ASSIGN chCtrlFrame = CtrlFrame:COM-HANDLE
         UIB_S = chCtrlFrame:LoadControls( OCXFile, "CtrlFrame":U).
  RUN initialize-controls IN THIS-PROCEDURE NO-ERROR.
END.
```



Design Time Properties in the AppBuilder

To customise the definition for a DashboardDesigner control, you must change the values of control properties in the AppBuilder at design time. The DashboardDesigner control supports properties that can be changed at both design time and run time, and some properties that can only be changed at runtime (*runtime properties*).

Setting Design Time Properties

The AppBuilder provides access to all available design time properties using the OCX Property Editor window. This window contains all design time properties, including the extended properties that Progress adds. Follow these steps in the AppBuilder to open the OCX Property Editor:

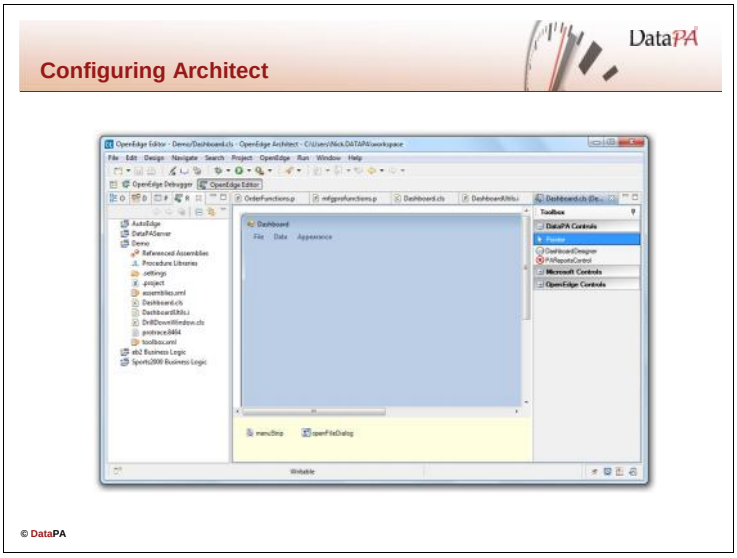
- Open a container object that contains a DashboardDesigner object.
- Double-click on the DashboardDesigner object.

The properties available in the OCX Property Editor window are described in the table below:

Property Name	Type	Definition
AboutMenuItemLabel	String	The title of the about menu item in the context menu that displays when a user right-clicks on a chart in the dashboard.
AboutMenuItemLink	String	The link that will be opened when the user selects the about menu item in the context menu that displays when a user right-clicks on a chart in the dashboard.
BackgroundColor	Integer	The background colour of the control. The value should be a colour defined by a hexadecimal colour in the format RRGGBB, converted to an integer.
DebugLogging	Boolean	If set to true, certain internal processes will be logged in the file [USER_TEMP_DIR]\DataPAEnterpriseDashboard.log.

Property Name	Type	Definition
DelayedRender	Boolean	The rendering of the dashboard will be delayed until a query has been refreshed if it is opened with the OpenDashboardDelayedRender method. The DelayedRender property will be true if the rendering of the open dashboard has been delayed. If the dashboard has been opened with the OpenDashboardDelayedRender method you can force it to be rendered without refreshing the query by setting the DelayedRender property to false.
Enabled	Boolean	Indicates whether the control is enabled to accept user input.
ForegroundColor	Integer	The default foreground colour for the control. The value should be a colour defined by a hexadecimal colour in the format RRGGBB, converted to an integer.
FullScreenMode	Boolean	Determines whether the control is rendered to support full screen. Setting this property to true does not change the size or location of the control, it merely determines what is displayed on the control. In order to render the dashboard full screen, it is the developers responsibility to hide the border and other controls on the form and resize the dashboard control.
HonorProKeys	logical	Default is TRUE. Determines who processes the GO, ENDKEY, HELP, and TAB keys: Progress, or the DashboardDesigner control. If the property is TRUE, Progress intercepts these keys and processes them as normal Progress key events. If the property is FALSE, the keystrokes are sent to the DashboardDesigner control for processing.
HonorReturnKey	logical	Default is FALSE. If the property is TRUE, Progress intercepts the key and processes it as a normal Progress RETURN key event. If the property is FALSE, which is the default, the keystroke is sent to the DashboardDesigner control for processing.
Name	string	The Name property contains the name of the control. The name is important because it identifies the control. You can use the control's name to get a COM-HANDLE to the control (for example, ASSIGN chDashboard = chCtrlFrame:DashboardDesigner, where DashboardDesigner is the control's name and chCtrlFrame is the control frame handle). The control name associates event handlers with a control.
PaletteMode	integer	Gets or sets palette style that is applied to the dashboard control to change its and any child forms appearance. Valid values are: ProfessionalSystem = 70280 ProfessionalOffice2003 = 70281 Office2007Blue = 70282 Office2007Silver = 70283 Office2007Black = 70284 Office2010Silver = 70286 Office2010Black = 70287 SparkleBlue = 70288 SparkleOrange = 70289 SparklePurple = 70290 Office2010Blue = 70285

Property Name	Type	Definition
ShowAboutMenu	Boolean	Determines if the about menu item appears in the context menu that displays when a user right-clicks on a chart in the dashboard.
ShowDataViewersWhenViewerOnly	Boolean	Determines if the data viewers that display the raw data returned by the query are accessible when the ViewerOnly property is set to true. The default value for this property is false.
ShowTabs	Boolean	Determines if the control displays the tabs that are used to select the different dashboard tabs in the control. The default is true.
StatusBarVisible	Boolean	Determines if the status bar is visible at the bottom of the control. The default is true.
Tag	String	The Tag property is a user property that allows the user to store an arbitrary string value and retrieve it later. Progress does not use this property internally, and it is intended to give the user a way of storing application specific information with the control. This property is initialized to an empty string.
ViewerOnly	Boolean	If set to True, the Dashboard control will act as a viewer only, disabling and preventing the display of any controls associated with editing a dashboard. The default is false, however if the license installed on the client does not support editing of the dashboard, this value will be true and cannot be changed.
Visible	Logical	The Visible property determines and indicates whether a PAReports control is currently displayed. The Visible property is distinct from, but influenced by, the Visible and Hidden attributes of the Control-Frame widget. The Visible property will appear in the Property Editor and can be set at design time. It defaults to TRUE. The value set in the Property Editor determines whether the OCX is initially displayed when the program is run, but can be overridden by the value of the Control-Frame widget's Hidden attribute.

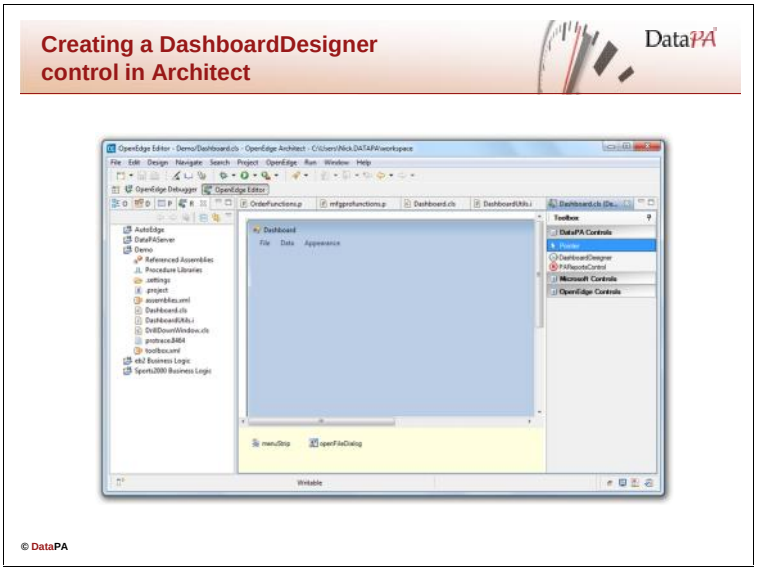


Configuring Architect

To provide quick and simple access to the DataPA DashboardDesigner Control in the Architect, we can add it to one of the control groups in the Toolbox.

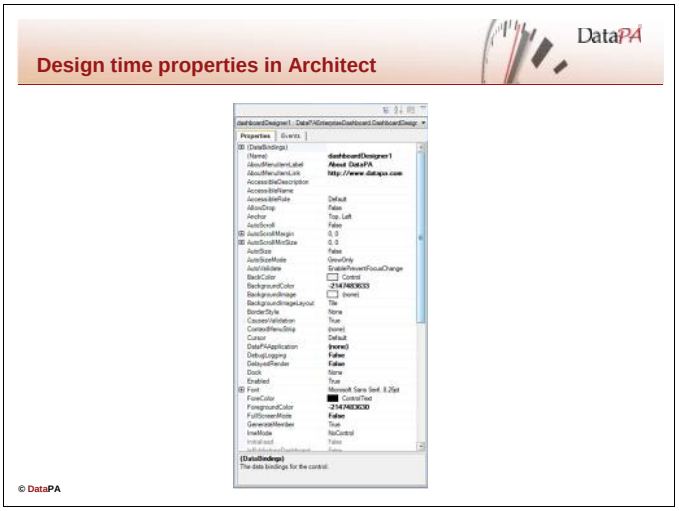
Follow these steps to add the DataPA Report Control to the Toolbox.

- Start a copy of the OpenEdge Architect
- Create a new OpenEdge Project
- In the Resources Treeview select the newly created project and right click
- Select New and then ABL Form
- Right click on the toolbox and select Add Controls
- Click on the Global Assemblies tab
- Check the DashboardDesigner assembly from the list and select OK
- The DashboardDesigner controls should now be available in the Toolbox.



Creating a DashboardDesigner Instance in the Architect

To add an instance DataPA DashboardDesigner control to your form select the DashboardDesigner toolbox item added to the Toolbox in Architect earlier in this lesson and draw the control onto the form.



Design Time Properties in Architect

To customise the definition for a DashboardDesigner control, you must change the values of control properties in the Architect at design time. The DashboardDesigner control supports properties that can be changed at both design time and run time, and some properties that can only be changed at runtime (*runtime properties*).

Setting Design Time Properties

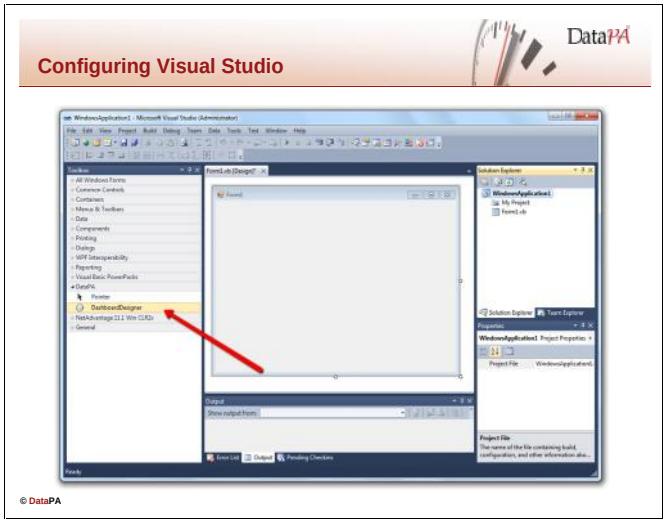
Architect provides access to all available design time properties using the Toolbox. This window contains all design time properties, including the extended properties that Architect adds.

Those properties available in the Toolbox window that are either specific or pertinent to the DashboardDesigner are described in the table below:

Property Name	Type	Definition
AboutMenuItemLabel	String	The title of the about menu item in the context menu that displays when a user right-clicks on a chart in the dashboard.
AboutMenuItemLink	String	The link that will be opened when the user selects the about menu item in the context menu that displays when a user right-clicks on a chart in the dashboard.
BackgroundColor	Integer	The background colour of the control. The value should be a colour defined by a hexadecimal colour in the format RRGGBB, converted to an integer.
DataPAApplication	DataPA.Application	The DataPA Application Object used to provide the core DataPA data functionality. If this property has not been assigned a valid DataPA Application object before the initialise method is called, the control will create its own instance of the DataPA Application Object.

Property Name	Type	Definition
DebugLogging	Boolean	If set to true, certain internal processes will be logged in the file [USER_TEMP_DIR]\DataPAEnterpriseDashboard.log.
DelayedRender	Boolean	The rendering of the dashboard will be delayed until a query has been refreshed if it is opened with the OpenDashboardDelayedRender method. The DelayedRender property will be true if the rendering of the open dashboard has been delayed. If the dashboard has been opened with the OpenDashboardDelayedRender method you can force it to be rendered without refreshing the query by setting the DelayedRender property to false.
Dock	DockStyle	Defines which borders of the control are bound to the container
Enabled	Boolean	Indicates whether the control is enabled to accept user input.
ForegroundColor	Integer	The default foreground colour for the control. The value should be a colour defined by a hexadecimal colour in the format RRGGBB, converted to an integer.
FullScreenMode	Boolean	Determines whether the control is rendered to support full screen. Setting this property to true does not change the size or location of the control, it merely determines what is displayed on the control. In order to render the dashboard full screen, it is the developers responsibility to hide the border and other controls on the form and resize the dashboard control.
Name	string	The name used to identify this control in the code.
PaletteMode	integer	Gets or sets palette style that is applied to the dashboard control to change its and any child forms appearance. Valid values are: ProfessionalSystem = 70280 ProfessionalOffice2003 = 70281 Office2007Blue = 70282 Office2007Silver = 70283 Office2007Black = 70284 Office2010Silver = 70286 Office2010Black = 70287 SparkleBlue = 70288 SparkleOrange = 70289 SparklePurple = 70290 Office2010Blue = 70285

Property Name	Type	Definition
SavedLoginDetails	Collection	Gets and sets a collection that contains a username and password that will be used as the default properties to connect to a server. The collection should contain two items... 1. A string value containing the username with the key username. 2. A string value containing the password with the key password.
ShowAboutMenu	Boolean	Determines if the about menu item appears in the context menu that displays when a user right-clicks on a chart in the dashboard.
ShowDataViewersWhenViewerOnly	Boolean	Determines if the data viewers that display the raw data returned by the query are accessible when the ViewerOnly property is set to true. The default value for this property is false.
ShowTabs	Boolean	Determines if the control displays the tabs that are used to select the different dashboard tabs in the control. The default is true.
StatusBarVisible	Boolean	Determines if the status bar is visible at the bottom of the control. The default is true.
Tag	String	The Tag property is a user property that allows the user to store an arbitrary string value and retrieve it later.
ViewerOnly	Boolean	If set to True, the Dashboard control will act as a viewer only, disabling and preventing the display of any controls associated with editing a dashboard. The default is false, however if the license installed on the client does not support editing of the dashboard, this value will be true and cannot be changed.
Visible	Logical	Determines if the control is visible in the form.

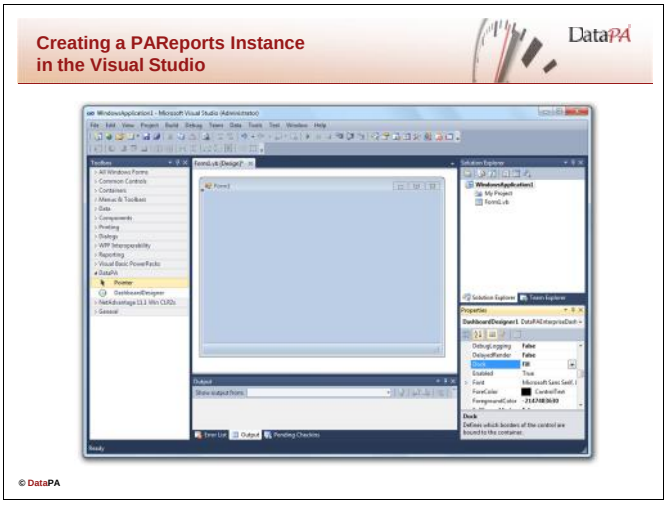


Configuring Visual Studio

To provide quick and simple access to the DataPA DashboardDesigner Control in the Visual Studio, we can add it to Toolbox.

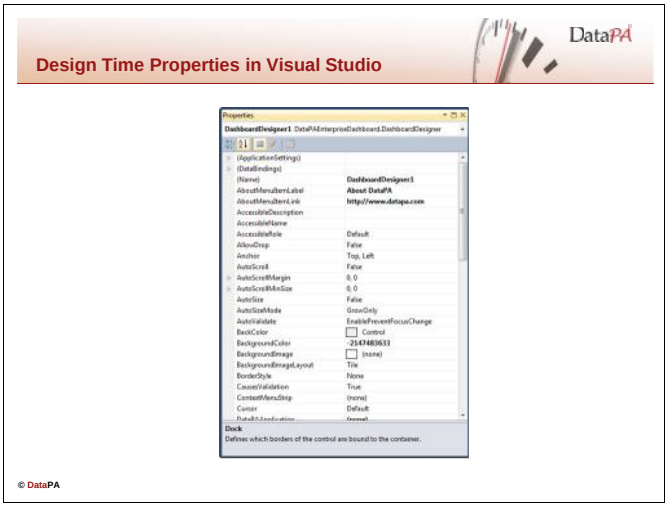
Follow these steps to add the DataPA DashboardDesigner Control to the Toolbox.

- Start a copy of the Microsoft Visual Studio
- Create a new Windows Forms application
- Right click on the Toolbox and select Choose Items.
- Then check the DashboardDesigner item from the list on the .NET Framework Components tab and click OK.



Creating a DashboardDesigner Instance in the Visual Studio

To add an instance DataPA DashboardDesigner control to your form select the DashboardDesigner item that was added to the Toolbox in Visual Studio earlier in this lesson and draw the control onto the form.



Design Time Properties in the Visual Studio

To customise the definition for a DashboardDesigner control, you must change the values of control properties in the Visual Studio at design time. The DashboardDesigner control supports properties that can be changed at both design time and run time, and some properties that can only be changed at runtime (*runtime properties*).

Setting Design Time Properties

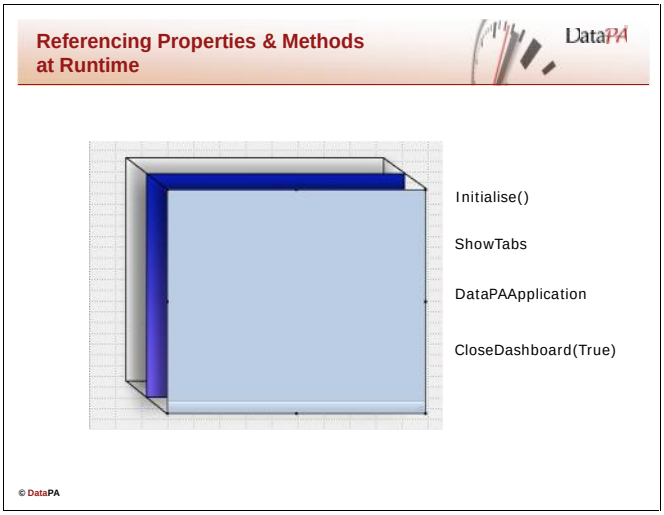
Visual Studio provides access to all available design time properties using the Properties Toolbox. This window contains all design time properties, including the extended properties that Visual Studio adds.

Those properties available in the Toolbox window that are either specific or pertinent to the DashboardDesigner are described in the table below:

Property Name	Type	Definition
AboutMenuItemLabel	String	The title of the about menu item that shows up in the context menu that displays when a user right-clicks on a chart in the dashboard.
AboutMenuItemLink	String	The link that will be opened when the user selects the about menu item that shows up in the context menu that displays when a user right-clicks on a chart in the dashboard.
BackgroundColor	Integer	The background colour of the control. The value should be a colour defined by a hexadecimal colour in the format RRGGBB, converted to an integer.
DataPAApplication	DataPA.Application	The DataPA Application Object used to provide the core DataPA data functionality. If this property has not been assigned a valid DataPA Application object before the initialise method is called, the control will create its own instance of the DataPA Application Object.

Property Name	Type	Definition
DebugLogging	Boolean	If set to true, certain internal processes will be logged in the file [USER_TEMP_DIR]\DataPAEnterpriseDashboard.log.
DelayedRender	Boolean	The rendering of the dashboard will be delayed until a query has been refreshed if it is opened with the OpenDashboardDelayedRender method. The DelayedRender property will be true if the rendering of the open dashboard has been delayed. If the dashboard has been opened with the OpenDashboardDelayedRender method you can force it to be rendered without refreshing the query by setting the DelayedRender property to false.
Dock	DockStyle	Defines which borders of the control are bound to the container
Enabled	Boolean	Indicates whether the control is enabled to accept user input.
ForegroundColor	Integer	The default foreground colour for the control. The value should be a colour defined by a hexadecimal colour in the format RRGGBB, converted to an integer.
FullScreenMode	Boolean	Determines whether the control is rendered to support full screen. Setting this property to true does not change the size or location of the control, it merely determines what is displayed on the control. In order to render the dashboard full screen, it is the developers responsibility to hide the border and other controls on the form and resize the dashboard control.
Name	string	The name used to identify this control in the code.
PaletteMode	integer	Gets or sets palette style that is applied to the dashboard control to change its and any child forms appearance. Valid values are: ProfessionalSystem = 70280 ProfessionalOffice2003 = 70281 Office2007Blue = 70282 Office2007Silver = 70283 Office2007Black = 70284 Office2010Silver = 70286 Office2010Black = 70287 SparkleBlue = 70288 SparkleOrange = 70289 SparklePurple = 70290 Office2010Blue = 70285

Property Name	Type	Definition
SavedLoginDetails	Collection	Gets and sets a collection that contains a username and password that will be used as the default properties to connect to a server. The collection should contain two items... 1. A string value containing the username with the key username. 2. A string value containing the password with the key password.
ShowAboutMenu	Boolean	Determines if the about menu item appears in the context menu that displays when a user right-clicks on a chart in the dashboard.
ShowDataViewersWhenViewerOnly	Boolean	Determines if the data viewers that display the raw data returned by the query are accessible when the ViewerOnly property is set to true. The default value for this property is false.
ShowTabs	Boolean	Determines if the control displays the tabs that are used to select the different dashboard tabs in the control. The default is true.
StatusBarVisible	Boolean	Determines if the status bar is visible at the bottom of the control. The default is true.
Tag	String	The Tag property is a user property that allows the user to store an arbitrary string value and retrieve it later.
ViewerOnly	Boolean	If set to True, the Dashboard control will act as a viewer only, disabling and preventing the display of any controls associated with editing a dashboard. The default is false, however if the license installed on the client does not support editing of the dashboard, this value will be true and cannot be changed.
Visible	Logical	Determines if the control is visible in the form.



Referencing Properties & Methods at Runtime

The syntax to access properties and methods at runtime largely depends on which of the three development environments you are using. The following section gives examples for each of the three supported developer environments:

AppBuilder Property Syntax

The syntax for referencing a property of the DashboardDesigner COM object is:

```
COMhdl-expression:Property-Name-Reference
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

Property-Name-Reference
Specifies a single DashboardDesigner property.

The following example assigns a value of false to the ShowTabs property of our DashboardDesigner object:

```
DashboardDesigner:ShowTabs = False.
```

Visual Studio Property Syntax

The syntax for referencing a property of the DashboardDesigner object is:

```
Control.Property-Name-Reference
```

Control
Is an expression that returns a component handle of a realised DashboardDesigner instance.

Property-Name-Reference
Specifies a single DashboardDesigner property.

The following example assigns a value of false to the ShowTabs property of our DashboardDesigner object:

```
DashboardDesigner.ShowTabs = False
```

Architect Property Syntax

The syntax for referencing a property of the DashboardDesigner object is:

```
Control:Property-Name-Reference
```

Control
Is an expression that returns a component handle of a realised DashboardDesigner instance.

Property-Name-Reference
Specifies a single DashboardDesigner property.

The following example assigns a value of false to the ShowTabs property of our DashboardDesigner object:

```
DashboardDesigner:ShowTabs = False.
```

As mentioned above, this syntax can be extended to string together several properties to reference a single property. The following example uses the Dashboard property of the DashboardDesigner object, and the Tabs property of the Dashboard object to return the handle of the tabs used in the current dashboard:

AppBuilder Example

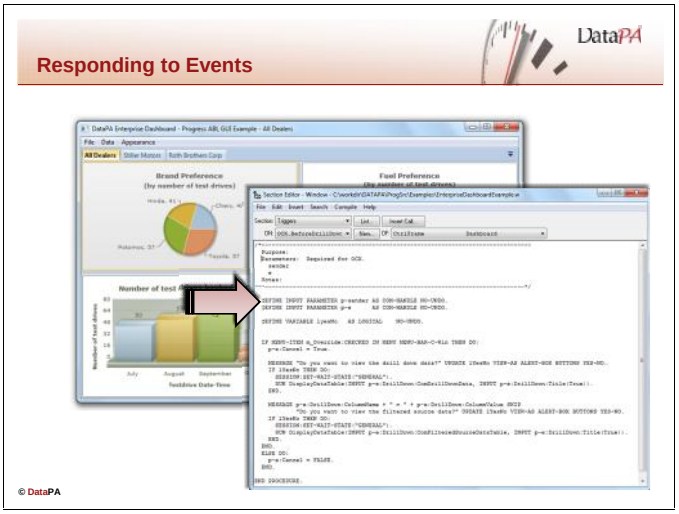
```
chTabs = DashboardDesigner:Dashboard:Tabs
```

Visual Studio Example

```
Tabs = DashboardDesigner.Dashboard.Tabs
```

Architect Example

```
Tabs = DashboardDesigner:Dashboard:Tabs.
```



Responding to Events in the AppBuilder

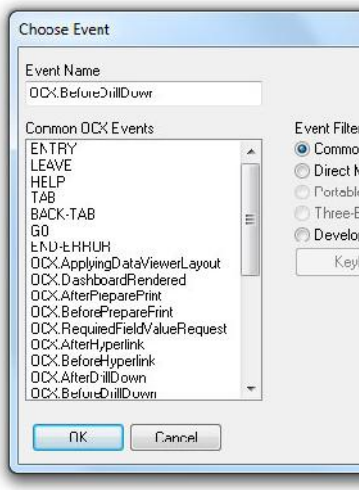
Progress handles DashboardDesigner events using OCX event procedures. An *OCX event procedure* is a standard Progress internal procedure that serves as an event handler for ActiveX controls. DashboardDesigner will often pass one or more parameters that must be handled by the event procedure. These parameters are defined as parameters in the event procedure. Progress identifies an OCX event procedure from the construction of its name. This is the only syntactic feature that distinguishes Progress internal procedures as an OCX event procedure.

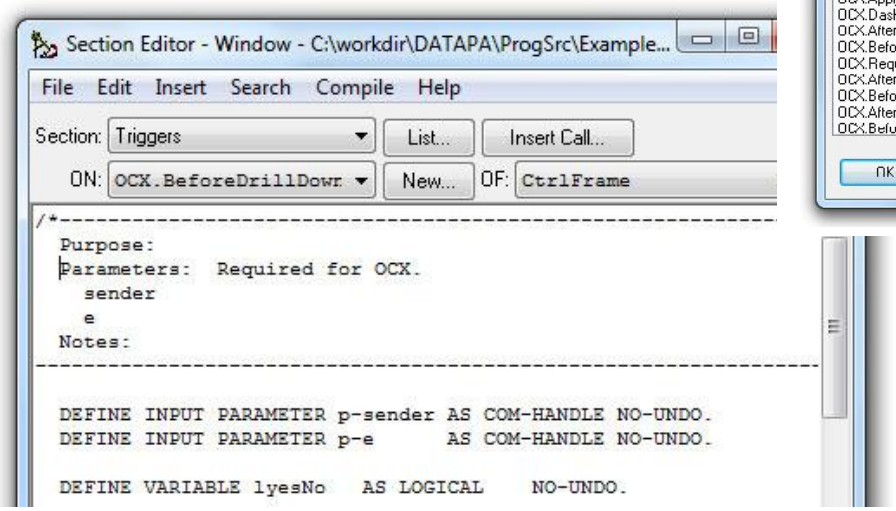
Creating Event Procedures in the AppBuilder

Follow these steps to create an OCX event procedure in the AppBuilder:

- Click on the edit code button in the AppBuilder to open the *section editor*
- Select *Triggers* in the *Section* combo-box
- Select the control frame in the *OF* combo-box
- Press New
- Select the appropriate event

The AppBuilder will create an appropriate event procedure with the relevant paramaters…





Responding to Events in Visual Studio

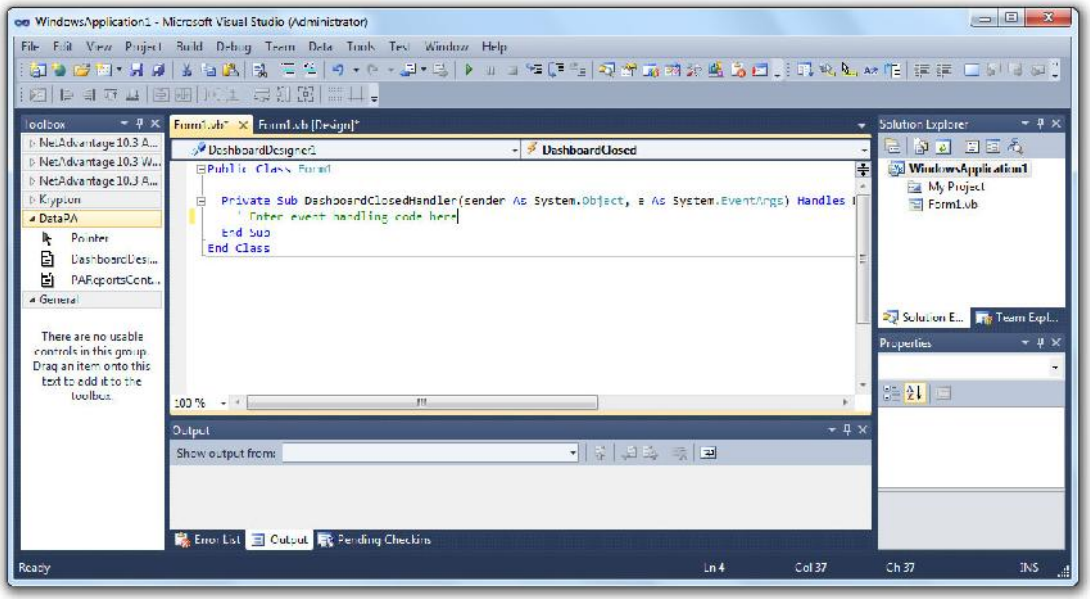
Visual Studio handles DashboardDesigner events using standard .NET event procedures. The DashboardDesigner will often pass one or more parameters that must be handled by the event procedure. These parameters are defined as parameters in the event procedure. Visual Studio identifies an event procedure from handler that is assigned to it, for example (`Handles DashboardDesigner1.DashboardClosed`).

Creating Event Procedures in Visual Studio

Follow these steps to create an event procedure in Visual Studio:

- Select the View Code option for the form which hosts the DashboardDesigner control
- Select the DashboardDesigner instance from the object drop down
- Select the event from the drop down list of events beside it

Visual Studio will create an appropriate event procedure with the correct handler assigned.



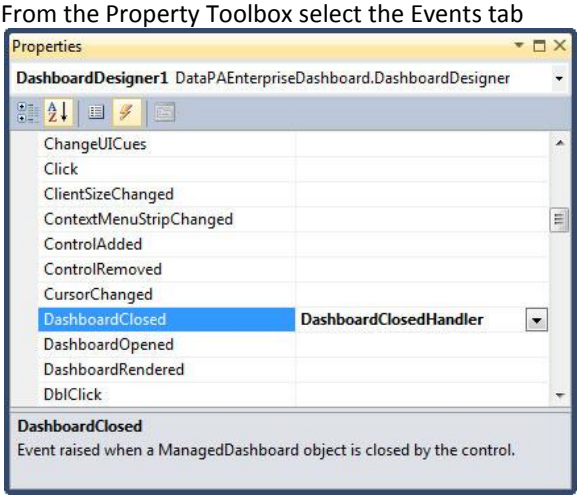
Responding to Events in Architect

Architect handles DashboardDesigner events using Architect event procedures. The DashboardDesigner will often pass one or more parameters that must be handled by the event procedure. These parameters are defined as parameters in the event procedure. Architect identifies an event procedure using the subscribe statement that is set in the InitializeComponent method of the ABL Form. For example (`THIS-OBJECT:DashboardDesigner:DashboardClosed:Subscribe(THIS-OBJECT:DashboardClosedHandler).`). The code to handle the event should then be placed in the method created by Architect.

Creating Event Procedures in Architect

Follow these steps to create an event procedure in Architect:

- Select the instance of DashboardDesigner in the ABL Form designer
- From the Property Toolbox select the Events tab



- Against the event you want to add event procedure for enter the name of the procedure you want to use.
- Architect will add a new method to the ABL Form which will have the name entered.

Initialising the DashboardDesigner Object

As we discussed in Lesson 2, the DashboardDesigner requires a DataPA Application object to provide the core functionality for DataPA. It is usually the case that you will want to share a single DataPA application object through all the instances of the DashboardDesigner and other DataPA controls. This not only improves performance, but will also prevent DataPA repeatedly connecting to an AppServer. When a DashboardDesigner control is created, it is on a non-initialised state, which means it does not respond to any user or code input. The control must be initialised, by calling the initialise method, before it can be used. If you do not assign a valid DataPA Application object to the DataPAApplication property of the control before calling the initialise method, the control will create its own DataPA Application object. The PAApplication property of the DashboardDesigner is described in detail in chapter 5.

The Initialise method

The Initialise method must be called before the DashboardDesigner control can be used.

AppBuilder Method Syntax

It is best practice to call the initialise method in the initialise-controls procedure in your Progress GUI window. The syntax for referencing this property of the DashboardDesigner COM object is:

```
COMhdl-expression:Initialise()
```

COMhdl-expression
Is an expression that returns a component handle of a realised ManagedDashboard instance.

The following example shows the use of the initialise method in our example application:

```
chDashboard:Initialise.
```

Visual Studio Method Syntax

It is best practice to call the initialise method in the load event of your .NET form. The syntax for referencing this property of the DashboardDesigner object is:

```
Control.Initialise()
```

Control

Is an expression that returns handle of a realised DashboardDesigner instance.

The following example shows the use of the initialise method in our example application:

```
        DashboardDesigner.Initialise()
```

Architect Method Syntax

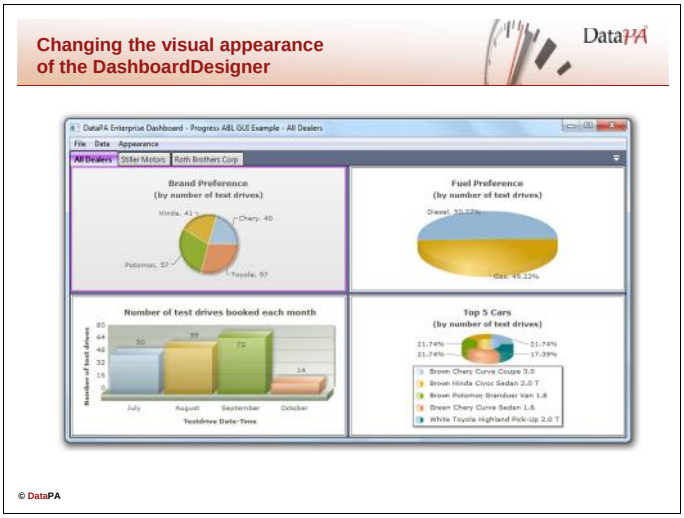
It is best practice to call the initialise method in the load event of your ABL .NET form. The syntax for referencing this property of the DashboardDesigner object is:

```
        Control:Initialise().
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

The following example shows the use of the initialise method in our example application:

```
        DashboardDesigner:Initialise().
```



Changing the visual appearance of the DashboardDesigner

There are a number of properties that you can use to manipulate the visual appearance and behaviour of the DashboardDesigner object to suit your application. The table below gives an overview of the properties which are described in detail below:

Property	Description
PaletteMode	Gets or sets palette style that is applied to the dashboard control to change its and any child forms appearance. Valid values are: ProfessionalSystem = 70280 ProfessionalOffice2003 = 70281 Office2007Blue = 70282 Office2007Silver = 70283 Office2007Black = 70284 Office2010Silver = 70286 Office2010Black = 70287 SparkleBlue = 70288 SparkleOrange = 70289 SparklePurple = 70290 Office2010Blue = 70285
ViewerOnly	If set to True, the Dashboard control will act as a viewer only, disabling and preventing the display of any controls associated with editing a dashboard. The default is false, however if the license installed on the client does not support editing of the dashboard, this value will be true and cannot be changed.
FullScreenMode	Determines whether the control is rendered to support full screen. Setting this property to true does not change the size or location of the control, it merely determines what is displayed on the control. In order to render the dashboard full screen, it is the developers responsibility to hide the border and other controls on the form and resize the dashboard control.
ShowTabs	Determines if the control displays the tabs that are used to select the different dashboard tabs in the control. The default is true.
StatusStripVisible	Determines if the status bar is visible at the bottom of the control. The default is true.

The PaletteMode property

The PaletteMode property sets palette style that is applied to the DashboardDesigner control to change its and any child forms appearance.

AppBuilder Property Syntax

The syntax for referencing this property of the DashboardDesigner COM object is:

```
COMhdl-expression:PaletteMode = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
Any valid Progress integer expression that resolves to one of the valid PaletteMode values.

For example, the following code sets the PaletteMode to ProfessionalSystem:

```
chDashboard:PaletteMode = 70280.
```

Visual Studio Property Syntax

The syntax for referencing this property of the ManagedDashboard control from Visual Studio is:

```
Control.PaletteMode = expression
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid PaletteMode value.

For example, the following code sets the PaletteMode to ProfessionalSystem:

```
DashboardDesigner.PaletteMode =
DataPAEnterpriseDashboard.DashboardDesigner.DashboardPalette.Professi
onalSystem
```

Architect Property Syntax

The syntax for referencing this property of the DashboardDesigner control from Architect is:

```
Control:PaletteMode = expression.
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid PaletteMode value.

For example, the following code sets the PaletteMode to ProfessionalSystem:

```
DashboardDesigner:PaletteMode =
DataPAEnterpriseDashboard.DashboardDesigner+DashboardPalette:Professi
onalSystem.
```

The ViewerOnly property

The ViewerOnly property determines whether the Dashboard control will act as a viewer only, that is disabling and preventing the display of any controls associated with editing a dashboard. The default is false, however if the license installed on the client does not support editing of the dashboard, this value will be true and cannot be changed.

AppBuilder Property Syntax

The syntax for referencing this property of the DashboardDesigner COM object is:

```
COMhdl-expression:ViewerOnly = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
Any valid Progress logical expression that resolves to True or False.

For example, the following code sets the ViewerOnly property to true:

```
chDashboard:ViewerOnly = True.
```

Visual Studio Property Syntax

The syntax for referencing this property of the ManagedDashboard control from Visual Studio is:

```
Control.ViewerOnly = expression
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid boolean value.

For example, the following code sets the ViewerOnly property to true:

```
DashboardDesigner.ViewerOnly = true
```

Architect Property Syntax

The syntax for referencing this property of the DashboardDesigner control from Architect is:

```
Control:ViewerOnly = expression.
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid logical value.

For example, the following code sets the ViewerOnly property to true:

```
DashboardDesigner:ViewerOnly = True.
```

The FullScreenMode property

The FullScreenMode property determines whether the control is rendered to support full screen. Setting this property to true does not change the size or location of the control, it merely determines what is displayed on the control. In order to render the dashboard full screen, it is the developer's responsibility to hide the border and other controls on the form and resize the dashboard control.

AppBuilder Property Syntax

The syntax for referencing this property of the DashboardDesigner COM object is:

```
COMhdl-expression:FullScreenMode = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
Any valid Progress logical expression that resolves to True or False.

For example, the following code sets the FullScreenMode property to true:

```
chDashboard:FullScreenMode = True.
```

Visual Studio Property Syntax

The syntax for referencing this property of the ManagedDashboard control from Visual Studio is:

```
Control.FullScreenMode = expression
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid boolean value.

For example, the following code sets the FullScreenMode property to true:

```
DashboardDesigner.FullScreenMode = true
```

Architect Property Syntax

The syntax for referencing this property of the DashboardDesigner control from Architect is:

```
Control:FullScreenMode = expression.
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid logical value.

For example, the following code sets the FullScreenMode property to true:

```
DashboardDesigner:FullScreenMode = True.
```

The ShowTabs property

The ShowTabs property determines if the control displays the tabs that are used to select the different dashboard tabs in the control. The default is true.

AppBuilder Property Syntax

The syntax for referencing this property of the DashboardDesigner COM object is:

```
COMhdl-expression:ShowTabs = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
Any valid Progress logical expression that resolves to True or False.

For example, the following code sets the ShowTabs property to true:

```
chDashboard:ShowTabs = True.
```

Visual Studio Property Syntax

The syntax for referencing this property of the ManagedDashboard control from Visual Studio is:

```
Control.ShowTabs = expression
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid boolean value.

For example, the following code sets the ShowTabs property to true:

```
DashboardDesigner.ShowTabs = true
```

Architect Property Syntax

The syntax for referencing this property of the DashboardDesigner control from Architect is:

```
Control:ShowTabs = expression.
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid logical value.

For example, the following code sets the ShowTabs property to true:

```
DashboardDesigner:ShowTabs = True.
```

The StatusStripVisible property

The StatusStripVisible property determines if the status bar is visible at the bottom of the control. The default is true.

AppBuilder Property Syntax

The syntax for referencing this property of the DashboardDesigner COM object is:

```
COMhdl-expression:StatusStripVisible = expression
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
Any valid Progress logical expression that resolves to True or False.

For example, the following code sets the StatusStripVisible property to true:

```
chDashboard:StatusStripVisible = True.
```

Visual Studio Property Syntax

The syntax for referencing this property of the ManagedDashboard control from Visual Studio is:

```
Control.StatusStripVisible = expression
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid boolean value.

For example, the following code sets the StatusStripVisible property to true:

```
DashboardDesigner.StatusStripVisible = true
```

Architect Property Syntax

The syntax for referencing this property of the DashboardDesigner control from Architect is:

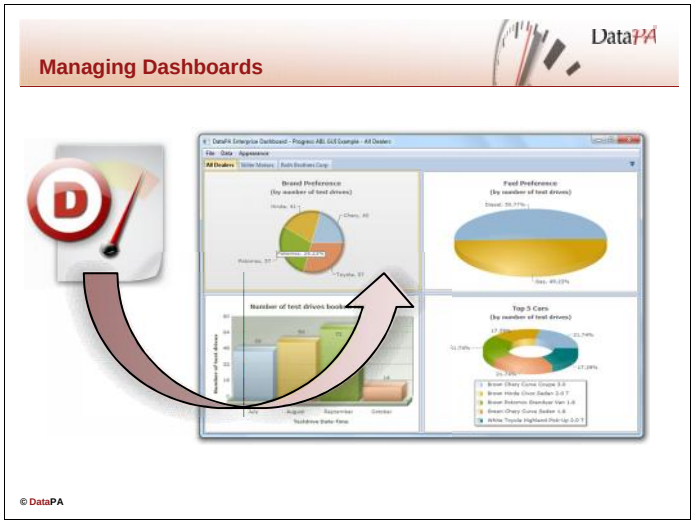
```
Control:StatusStripVisible = expression.
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
Any valid logical value.

For example, the following code sets the StatusStripVisible property to true:

```
DashboardDesigner:StatusStripVisible = True.
```



Managing Dashboards

There are a number of properties and methods that you can use to manage the dashboard that is currently open in the DashboardDesigner control. The table below gives an overview of the properties and methods, which are described in detail below:

Property/Method	Description
Dashboard	Retrieves the ManagedDashboard object that represents the dashboard currently open in the control.
OpenDashboard	Opens a dashboard from disk.
OpenDashboardDelayedRender	Opens a ManagedDashboard object from disk and delays the rendering of the dashboard until after a query is refreshed.
CreateDashboard	Create a new ManagedDashboard object and opens it in the control.
SaveDashboard	Saves the current ManagedDashboard object to disk.
CloseDashboard	Closes the currently open ManagedDashboard object. Accepts a single boolean parameter force. If force is TRUE, the user will not be prompted to save any changes.
RefreshQueries	Refreshes all the queries in the current ManagedDashboard.
Print	Prints the current ManagedDashboard object.
PrintPreview	Opens the print preview dialog for the current ManagedDashboard object.

The Dashboard property

The Dashboard property on the DashboardDesigner control provides access to the ManagedDashboard object that represents the dashboard that is currently open in the control.

AppBuilder Property Syntax

The syntax for referencing this property of the DashboardDesigner COM object is:

```
expression  = COMhdl-expression:Dashboard
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
A COM-HANDLE variable.

For example, the following code gets a handle to the ManagedDashboard object:

```
chDashboard = chDashboardDesigner:Dashboard.
```

Visual Studio Property Syntax

The syntax for referencing this property of the ManagedDashboard control from Visual Studio is:

```
expression  = Control.Dashboard
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A DataPAEnterpriseDashboard.ManagedDashboard variable.

For example, the following code gets a handle to the ManagedDashboard object:

```
Dashboard = DashboardDesigner.Dashboard
```

Architect Property Syntax

The syntax for referencing this property of the DashboardDesigner control from Architect is:

```
expression  = Control:Dashboard.
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A DataPAEnterpriseDashboard.ManagedDashboard variable.

For example, the following code gets a handle to the ManagedDashboard object:

```
Dashboard = DashboardDesigner:Dashboard.
```

The OpenDashboard method

The OpenDashboard method allows you to open a dashboard in the control from disk.

AppBuilder Property Syntax

The syntax for referencing this method of the DashboardDesigner COM object is:

COMhdl-expression:OpenDashboard(*expression)*

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
A String expression that contains the name and location of the dashboard file on disk.

For example, the following code opens a ManagedDashboard object:

```
chDashboardDesigner:OpenDashboard("C:\myDashboard.edp").
```

Visual Studio Property Syntax

The syntax for referencing this method of the ManagedDashboard control from Visual Studio is:

Control.OpenDashboard(*expression)*

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A String expression that contains the name and location of the dashboard file on disk.

For example, the following code opens a ManagedDashboard object:

```
DashboardDesigner.OpenDashboard("C:\myDashboard.edp")
```

Architect Property Syntax

The syntax for referencing this method of the DashboardDesigner control from Architect is:

Control:OpenDashboard(*expression*).

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A String expression that contains the name and location of the dashboard file on disk.

For example, the following code opens a ManagedDashboard object:

```
DashboardDesigner:OpenDashboard("C:\myDashboard.edp").
```

The OpenDashboardDelayedRender method

The OpenDashboardDelayedRender method allows you to open a dashboard in the control and delays the rendering of the dashboard until after a query is refreshed. This is useful when you want to ensure only the latest information is displayed in the dashboard.

AppBuilder Property Syntax

The syntax for referencing this method of the DashboardDesigner COM object is:

COMhdl-expression:OpenDashboardDelayedRender(*expression)*

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
A String expression that contains the name and location of the dashboard file on disk.

For example, the following code opens a ManagedDashboard object:

```
chDashboardDesigner:OpenDashboardDelayedRender("C:\myDashboard.edp").
```

Visual Studio Property Syntax

The syntax for referencing this method of the ManagedDashboard control from Visual Studio is:

Control.OpenDashboardDelayedRender(*expression)*

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A String expression that contains the name and location of the dashboard file on disk.

For example, the following code opens a ManagedDashboard object:

```
DashboardDesigner.OpenDashboardDelayedRender("C:\myDashboard.edp")
```

Architect Property Syntax

The syntax for referencing this method of the DashboardDesigner control from Architect is:

Control:DashboardDelayedRender(*expression*).

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A String expression that contains the name and location of the dashboard file on disk.

For example, the following code opens a ManagedDashboard object:

```
DashboardDesigner:DashboardDelayedRender("C:\myDashboard.edp").
```

The CreateDashboard method

The CreateDashboard method creates a new dashboard and renders it in the control.

AppBuilder Property Syntax

The syntax for referencing this method of the DashboardDesigner COM object is:

```
COMhdl-expression:CreateDashboard()
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

For example, the following code creates a ManagedDashboard object:

```
chDashboardDesigner:CreateDashboard().
```

Visual Studio Property Syntax

The syntax for referencing this method of the ManagedDashboard control from Visual Studio is:

```
Control.CreateDashboard()
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

For example, the following code creates a ManagedDashboard object:

```
DashboardDesigner.CreateDashboard()
```

Architect Property Syntax

The syntax for referencing this method of the DashboardDesigner control from Architect is:

```
Control:CreateDashboard().
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

For example, the following code opens a ManagedDashboard object:

```
DashboardDesigner:CreateDashboard().
```

The SaveDashboard method

The SaveDashboard method allows you to save a dashboard in the control to disk.

AppBuilder Property Syntax

The syntax for referencing this method of the DashboardDesigner COM object is:

COMhdl-expression:SaveDashboard(*expression)*

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
A Boolean expression that indicates if the SaveAs dialog should be used regardless of whether the dashboard has been saved before.

For example, the following code saves a ManagedDashboard object:

```
chDashboardDesigner:SaveDashboard(False).
```

Visual Studio Property Syntax

The syntax for referencing this method of the ManagedDashboard control from Visual Studio is:

Control.SaveDashboard(*expression)*

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A Boolean expression that indicates if the SaveAs dialog should be used regardless of whether the dashboard has been saved before.

For example, the following code saves a ManagedDashboard object:

```
DashboardDesigner.SaveDashboard(False)
```

Architect Property Syntax

The syntax for referencing this method of the DashboardDesigner control from Architect is:

Control:SaveDashboard(*expression*).

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A Boolean expression that indicates if the SaveAs dialog should be used regardless of whether the dashboard has been saved before.

For example, the following code saves a ManagedDashboard object:

```
DashboardDesigner:SaveDashboard(False).
```

The CloseDashboard method

The CloseDashboard method allows you to close a dashboard currently open in the control.

AppBuilder Property Syntax

The syntax for referencing this method of the DashboardDesigner COM object is:

```
COMhdl-expression:CloseDashboard(expression)
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
A Boolean expression that if TRUE, the user will not be prompted to save any changes.

For example, the following code closes the open ManagedDashboard object without prompting the user to save any changes:

```
chDashboardDesigner:CloseDashboard(True).
```

Visual Studio Property Syntax

The syntax for referencing this method of the ManagedDashboard control from Visual Studio is:

```
Control.CloseDashboard(expression)
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A Boolean expression that if TRUE, the user will not be prompted to save any changes.

For example, the following code closes the open ManagedDashboard object without prompting the user to save any changes:

```
DashboardDesigner.CloseDashboard(True)
```

Architect Property Syntax

The syntax for referencing this method of the DashboardDesigner control from Architect is:

```
Control:CloseDashboard(expression).
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A Boolean expression that if TRUE, the user will not be prompted to save any changes.

For example, the following code closes the open ManagedDashboard object without prompting the user to save any changes:

```
DashboardDesigner:CloseDashboard(True).
```

The RefreshQueries method

The RefreshQueries method creates refreshes all the queries in the currently open ManagedDashboard control. The RefreshQueries method can be used to refresh the data, and cause the dashboard to be rendered after it has been opened with the OpendelayedRender method.

AppBuilder Property Syntax

The syntax for referencing this method of the DashboardDesigner COM object is:

```
COMhdl-expression:RefreshQueries()
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

For example, the following code refreshes the queries in a ManagedDashboard object:

```
chDashboardDesigner:RefreshQueries().
```

Visual Studio Property Syntax

The syntax for referencing this method of the ManagedDashboard control from Visual Studio is:

```
Control.RefreshQueries()
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

For example, the following code refreshes the queries in a ManagedDashboard object:

```
DashboardDesigner.RefreshQueries()
```

Architect Property Syntax

The syntax for referencing this method of the DashboardDesigner control from Architect is:

```
Control:RefreshQueries().
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

For example, the following code refreshes the queries in a ManagedDashboard object:

```
DashboardDesigner:RefreshQueries().
```

The Print method

The Print method allows you to print a dashboard currently open in the control.

AppBuilder Property Syntax

The syntax for referencing this method of the DashboardDesigner COM object is:

```
COMhdl-expression:Print(expression)
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

expression
A Boolean expression that indicates if the print dialog should be shown before the dashboard is printed.

For example, the following prints the open ManagedDashboard object without opening the print dialog:

```
chDashboardDesigner:Print(True).
```

Visual Studio Property Syntax

The syntax for referencing this method of the ManagedDashboard control from Visual Studio is:

```
Control.Print(expression)
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A Boolean expression that indicates if the print dialog should be shown before the dashboard is printed.

For example, the following prints the open ManagedDashboard object without opening the print dialog:

```
DashboardDesigner.Print(True)
```

Architect Property Syntax

The syntax for referencing this method of the DashboardDesigner control from Architect is:

```
Control:Print(expression).
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

expression
A Boolean expression that indicates if the print dialog should be shown before the dashboard is printed.

For example, the following prints the open ManagedDashboard object without opening the print dialog:

```
DashboardDesigner:Print(True).
```

The PrintPreview method

The PrintPreview method allows you to open the Print Preview dialog for the dashboard currently open in the control.

AppBuilder Property Syntax

The syntax for referencing this method of the DashboardDesigner COM object is:

```
COMhdl-expression:PrintPreview()
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

For example, the following opens the PrintPreview dialog for the open ManagedDashboard object:

```
chDashboardDesigner:PrintPreview().
```

Visual Studio Property Syntax

The syntax for referencing this method of the ManagedDashboard control from Visual Studio is:

```
Control.PrintPreview()
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

For example, the following opens the PrintPreview dialog for the open ManagedDashboard object:

```
DashboardDesigner.PrintPreview()
```

Architect Property Syntax

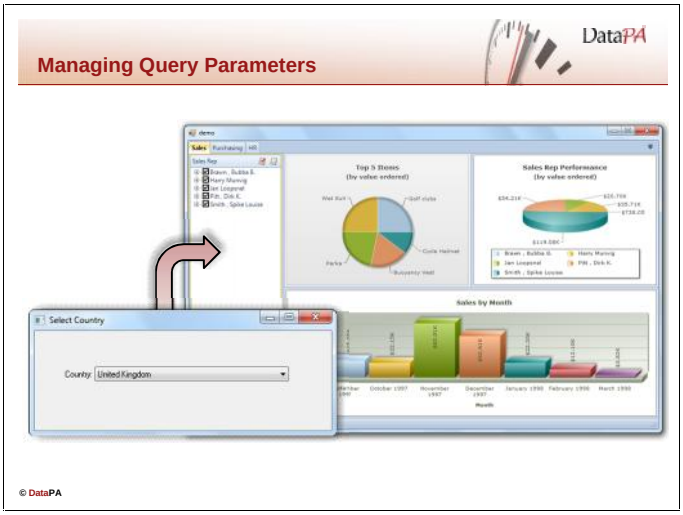
The syntax for referencing this method of the DashboardDesigner control from Architect is:

```
Control:PrintPreview().
```

Control
Is an expression that returns handle of a realised DashboardDesigner instance.

For example, the following opens the PrintPreview dialog for the open ManagedDashboard object:

```
DashboardDesigner:PrintPreview().
```



Managing Query Parameters

By default query parameters are added to a dashboard as a control panel object. The parameter values used to filter the query when it is refreshed are taken from these control panel objects, and the queries are refreshed when the user enters or selects a different value in this control.
However, when a dashboard is embedded into a third party application, it is often the case that you want to provide the query parameter values from the third party application, rather than the user controlling these values from an object on the dashboard.

Providing Query Parameters Values from an Application

When a query is refreshed in the dashboard, the DashboardDesigner control checks if there is a valid control panel object for each required field in the query. For each required field that does not have a valid control panel object, the DashboardDesigner fires an event to try and retrieve the query parameter value from the calling application.
Thus, to provide query parameter values from the third party application rather than from a control on the dashboard, follow these steps (each step is described in more detail later in the lesson):

1. Delete the control panel objects that provide query parameter values from the dashboard.
2. Provide code in your application to respond to the RequiredFieldValueRequest event to provide the parameter value for the query.

Deleting Control Panel Objects

Follow these steps to remove control panel objects that provide query parameter values from the dashboard.

1. Open the dashboard in the DataPA Enterprise Dashboard application.
2. Double click on the control panel in the dashboard to open the Control Panel dialog box.
3. Select the control panel object in the control panel.
4. Select the Delete button in the ribbon.
5. Press OK
6. Save the dashboard.

The RequiredFieldValueRequest Event in the AppBuilder

The RequiredFieldValueRequest event is raised by the DashboardDesigner control for each required field of a query that does not have a related control panel object when that query is run. It provides the developer with an opportunity to provide values for the required field.

Syntax for RequiredFieldValueRequest handler procedure

The syntax for responding to this event in the ManagedDashboard control from the AppBuilder is:

```
PROCEDURE COMhdl-expression.RequiredFieldValueRequest .

  DEFINE INPUT PARAMETER p-sender AS COM-HANDLE NO-UNDO.
  DEFINE INPUT PARAMETER p-e      AS COM-HANDLE NO-UNDO.

  [Code]
END PROCEDURE.
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

p-sender
Receives a handle to the DashboardDesigner that has raised the event.

p-e
Receives a handle to a RequiredFieldValueRequestEventArgs object that provides the interface to interact with the event.

Code
The code to respond to the event.

For example, the following code responds to the RequiredFieldValueRequest by applying United Kingdom to the country parameter :

```
PROCEDURE CtrlFrame.DashboardDesigner.RequiredFieldValueRequest .
  DEFINE INPUT PARAMETER p-sender AS COM-HANDLE NO-UNDO.
  DEFINE INPUT PARAMETER p-e      AS COM-HANDLE NO-UNDO.

  IF p-e:RequiredField:Name = "sports2000.Customer.Country" THEN DO:
    p-e:COMValue = "United Kingdom".
  END.

END PROCEDURE.
```

The RequiredFieldValueRequest Event in the Visual Studio

The RequiredFieldValueRequest event is raised by the DashboardDesigner control for each required field of a query that does not have a related control panel object when that query is run. It provides the developer with an opportunity to provide values for the required field.

Syntax for RequiredFieldValueRequest handler procedure

The syntax for responding to this event in the ManagedDashboard control from Visual Studio is:

```
Private Sub RequiredFieldValueRequestHandler(sender As Object, e As
DataPAEnterpriseDashboard.RequiredFieldValueRequestEventArgs)
Handles COMhdl-expression.RequiredFieldValueRequest
   [Code]
 End Sub
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

sender
Receives a handle to the DashboardDesigner that has raised the event.

e
Receives a handle to a RequiredFieldValueRequestEventArgs object that provides the interface to interact with the event.

Code
The code to respond to the event.

For example, the following code responds to the RequiredFieldValueRequest by applying United Kingdom to the country parameter :

```
Private Sub DashboardDesigner_RequiredFieldValueRequest(sender As
Object, e As
DataPAEnterpriseDashboard.RequiredFieldValueRequestEventArgs) Handles
DashboardDesigner.RequiredFieldValueRequest
    If e.RequiredField.Name = "sports2000.Customer.Country" Then
      e.Value = "United Kingdom"
    End If
  End Sub
```

The RequiredFieldValueRequest Event in the Architect

The RequiredFieldValueRequest event is raised by the DashboardDesigner control for each required field of a query that does not have a related control panel object when that query is run. It provides the developer with an opportunity to provide values for the required field.

Syntax for RequiredFieldValueRequest handler procedure

The syntax for responding to this event in the ManagedDashboard control from Architect is:

```
METHOD PRIVATE VOID QueryParamHandler( INPUT sender AS
System.Object, INPUT e AS
DataPAEnterpriseDashboard.RequiredFieldValueRequestEventArgs ):

 [Code]
END METHOD.
```

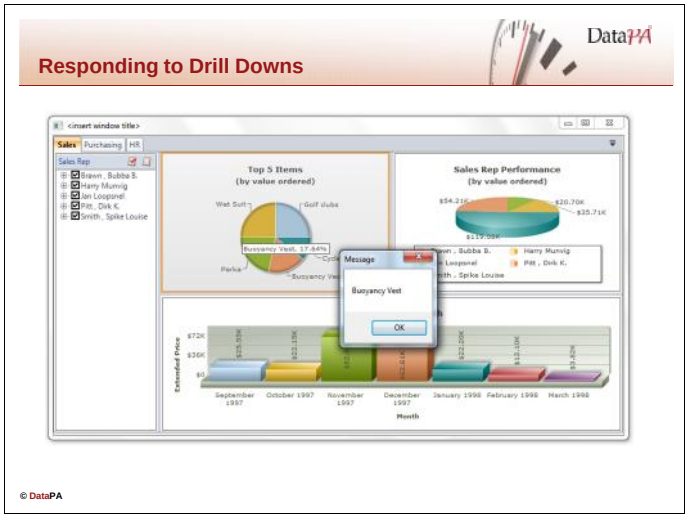
sender
Receives a handle to the DashboardDesigner that has raised the event.

e
Receives a handle to a RequiredFieldValueRequestEventArgs object that provides the interface to interact with the event.

Code
The code to respond to the event.

For example, the following code responds to the RequiredFieldValueRequest by applying United Kingdom to the country parameter :

```
METHOD PRIVATE VOID QueryParamHandler( INPUT sender AS System.Object,
INPUT e AS
DataPAEnterpriseDashboard.RequiredFieldValueRequestEventArgs ):
  IF e:RequiredField:Name = "sports2000.Customer.Country" THEN DO:
    e:Value = "United Kingdom".
  END.
  RETURN.
END METHOD.
```



Responding to Drill Downs

When embedded into a third party application, it is common to want the containing application to respond to the user clicking a specific area of a chart. The DashboardDesigner control has a BeforeDrillDown event which allows the containing application to respond to drill down events.

The BeforeDrillDown Event in the AppBuilder

The BeforeDrillDown event is raised by the DashboardDesigner when the user clicks on the sensitive area of a chart that has a drill down defined.

Syntax for BeforeDrillDown handler procedure

The syntax for referencing this method of the ManagedDashboard control from the AppBuilder is:

```
PROCEDURE COMhdl-expression.BeforeDrillDown .

  DEFINE INPUT PARAMETER p-sender AS COM-HANDLE NO-UNDO.
  DEFINE INPUT PARAMETER p-e      AS COM-HANDLE NO-UNDO.

  [Code]
END PROCEDURE.
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

p-sender
Receives a handle to the DashboardDesigner that has raised the event.

p-e
Receives a handle to a DrillDownEventArgs object that provides the interface to interact with the event.

Code
The code to respond to the event.

For example, the following code responds to the BeforeDrillDown by showing a message with the value selected:

```
PROCEDURE CtrlFrame.DashboardDesigner.BeforeDrillDown .
  DEFINE INPUT PARAMETER p-sender AS COM-HANDLE NO-UNDO.
  DEFINE INPUT PARAMETER p-e      AS COM-HANDLE NO-UNDO.


  MESSAGE p-e:DrillDown:ColumnValue VIEW-AS ALERT-BOX.
  p-e:Cancel = True.

END PROCEDURE..
```

The BeforeDrillDown Event in the Visual Studio

The BeforeDrillDown event is raised by the DashboardDesigner when the user clicks on the sensitive area of a chart that has a drill down defined.

Syntax for RequiredFieldValueRequest handler procedure

The syntax for responding to this event in the ManagedDashboard control from Visual Studio is:

```
Private Sub BeforeDrillDownHandler(sender As Object, e As
DataPAEnterpriseDashboard.DrillDownEventArgs) Handles COMhdl-
expression.BeforeDrillDown
  [Code]
 End Sub
```

COMhdl-expression
Is an expression that returns a component handle of a realised DashboardDesigner instance.

sender
Receives a handle to the DashboardDesigner that has raised the event.

e
Receives a handle to a DrillDownEventArgs object that provides the interface to interact with the event.

Code
The code to respond to the event.

For example, the following code responds to the BeforeDrillDown by showing a message with the value selected:

```
Private Sub BeforeDrillDownHandler(sender As System.Object, e As
DataPAEnterpriseDashboard.DrillDownEventArgs) Handles
DashboardDesigner.BeforeDrillDown
  MsgBox(e.DrillDown.ColumnValue)
End Sub
```

The BeforeDrillDown Event in the Architect

The BeforeDrillDown event is raised by the DashboardDesigner when the user clicks on the sensitive area of a chart that has a drill down defined.

Syntax for RequiredFieldValueRequest handler procedure

The syntax for responding to this event in the ManagedDashboard control from Architect is:

```
METHOD PRIVATE VOID QueryParamHandler( INPUT sender AS
System.Object, INPUT e AS
DataPAEnterpriseDashboard.DrillDownEventArgs ):

  [Code]
END METHOD.
```

sender
Receives a handle to the DashboardDesigner that has raised the event.

e
Receives a handle to a DrillDownEventArgs object that provides the interface to interact with the event.

Code
The code to respond to the event.

For example, the following code responds to the BeforeDrillDown by showing a message with the value selected:

```
METHOD PRIVATE VOID DrillDownHandler( INPUT sender AS System.Object,
INPUT e AS DataPAEnterpriseDashboard.DrillDownEventArgs ):
  MESSAGE e:DrillDown:ColumnValue VIEW-AS ALERT-BOX.
  e:Cancel = TRUE.
  RETURN.
END METHOD.
```

DataPA Application Object

Introduction

The DataPA Application objects are a set of objects that allow a developer to use DataPA to run queries and export the information into applications on the client machine.

Learning Objectives

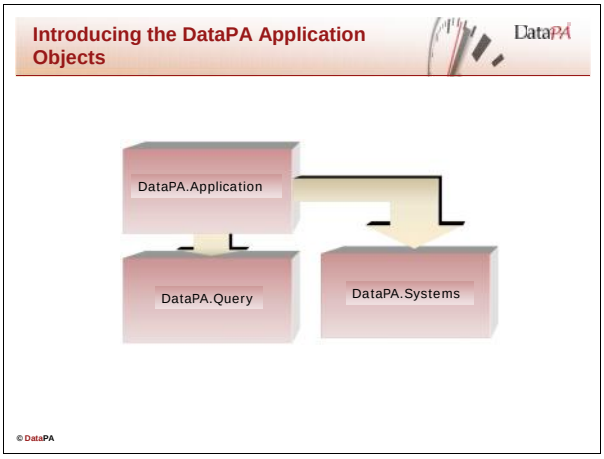
When you complete this lesson you should be able to:

- Understand the hierarchy of the DataPA Application Objects.
- Create and manage the DataPA Application Objects in an application.
- Use the DataPA Application Objects to export to Excel and Access.
- Respond to events from the DataPA Application Objects.
- Understand how to share the objects for increased performance.

Prerequisites

Before you begin this lesson you should be able to:

- Create a GUI application using the development tool of your choice.
- Use methods and properties on objects in the development tool.

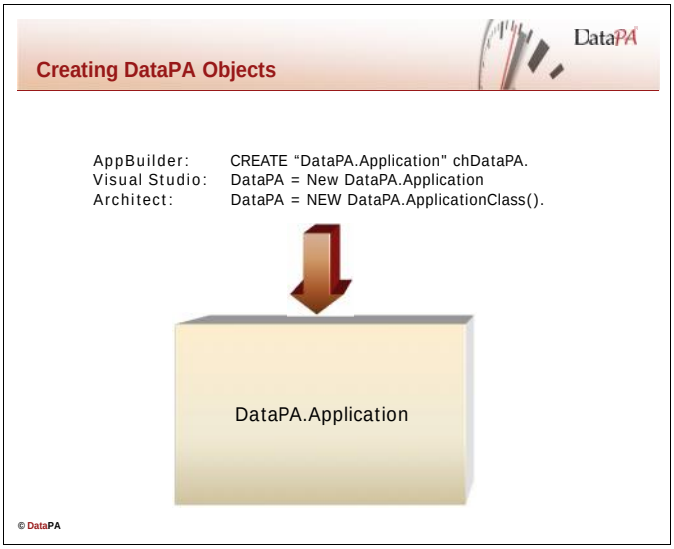


Introducing the DataPA Application Objects

DataPA provides a suite of Application objects to present a wide range of functionality to the end user. These objects can be used to run, create and modify queries.

The DataPA Application Objects

Object	Definition & Usage
DataPA.Application	The DataPA.Application object provides the core functionality for DataPA and is used for all of the DataPA interfaces. It provides the functionality to manage the Systems, Links and Subjects built by the administrator and used as the basis for all queries. It also provides the functionality to run queries. An event on the DataPA.Application object informs a containing procedure that data has been retrieved, presenting that data to be handled by the calling procedure.
DataPA.Query	The DataPA.Query object represents a single open query. It cannot be created by the developer, but is available from a property of the DataPA.Application object if a query is currently open. It provides properties and methods to allow the developer to manage the query and run the query.
DataPA.Systems	The DataPA.Systems object is a collection that provides access to each of the defined systems in DataPA. It cannot be created by the developer, but is available from a property of the DataPA.Application object. The DataPA.Systems object allows the developer to manage connection security to the AppServer and will be discussed in more details in Lesson 6: DataPA Security.



Creating DataPA Object

Like using the PAReports and DashboardDesigner objects, we must instantiate the objects before we can use them. Unlike the PAReports and DashboardDesigner objects, the DataPA Automation objects are non visual objects. As such, we cannot drag them from a toolbox or palette and so must create the code to instantiate them manually.

AppBuilder - The Create Automation Object Statement

The Create Automation object statement launches a new instance of an object and creates a connection to it. The syntax of the Create Automation Object statement is as follows:

```
CREATE expression COMhdl-expression
```

expression
A character string expression that evaluates to DataPA.Application

COMhdl-expression
A COM-HANDLE variable that receives the COM handle to the instantiated Automation object.

For example:

```
CREATE "DataPA.Application" chDataPA.
```

Visual Studio – Configuring to use the Application Object

If you have not added a DataPA Reports Control to the project already and want to use the DataPA Application Objects then you will need to add references to these manually before you can do so.

Follow these steps to add the appropriate references:

1. Go into Project properties and select the References tab.
2. Select the Browse tab and click the Add button.
3. Browse to the bin folder under the DataPA installation folder.
4. Add references to the following dll which is located in the bin folder under the DataPA installation folder:
   Interop.DataPA.dll
5. Click OK.

You should now be able to reference the DataPA Application objects.

The syntax of the creation of an object statement is as follows:

```
Objecthandle = expression
```

expression
A character string expression that evaluates to DataPA.Application object

Objecthandle
A Handle variable that receives the handle to the instantiated object.

For example:

```
DataPA = New DataPA.Application
```

Architect – Configuring to use the Application Object

If you have not added a DataPA Reports control or Enterprise Dashboard control to the project already and want to use the DataPA Application Objects then you will need to add references to these manually before you can do so.

Follow these steps to add the appropriate references:

1. Import the following dll's into your ABL project folder by selecting File and then Import from the menu in Architect. These files are located in the bin folder under the DataPA installation folder:
   Interop.ADODB.dll
   Interop.ADOR.dll
   Interop.DataPA.dll
   Interop.MSXML2.dll
   Interop.OutlookBar.dll
   Interop.RDS.dll
   Interop.SubclassingSink.dll
   Interop.VBA.dll
2. Add an ABL form to your project if you do not already have one.
3. Add a control to the ABL form if this has not already been done. This should make referenced assemblies node appear in resources treeview.
4. Right click on the referenced assemblies in the resources treeview and select add assembly reference option from the pop up menu.
5. Select the Local Assemblies tab and browse to the current project folder.
6. Add the dll's that were imported in step 1 above.

You should now be able to reference the DataPA Application objects.

The syntax of the creation of an object statement is as follows:

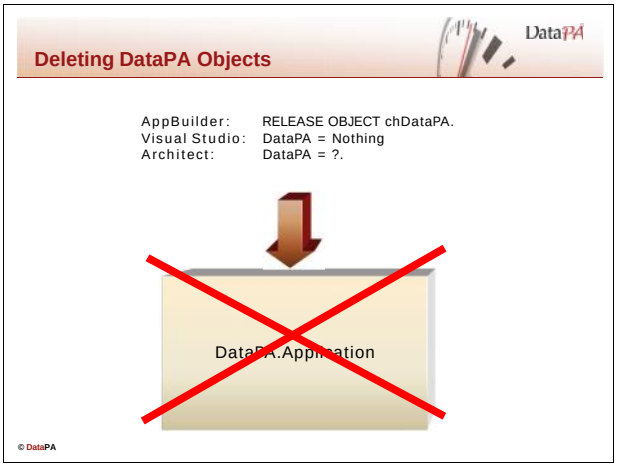
```
Objecthandle = expression
```

expression
A character string expression that evaluates to DataPA.Application object

Objecthandle
A Handle variable that receives the handle to the instantiated object.

For example:

```
DataPA = NEW DataPA.ApplicationClass().
```



Deleting DataPA Objects

Unlike static Progress widgets, DataPA Automation objects are scoped to the session. This means that if we do not explicitly delete the objects, they will stay resident in memory until the end of the session. This can lead to memory leaks, so we must ensure that we always delete objects once we have finished with them.

AppBuilder Syntax

The RELEASE Object statement destroys a DataPA automation object and releases the memory it occupied. The syntax for the RELEASE Object handle is as follows:

```
RELEASE OBJECT COM-hdl-var [ NO-ERROR ]
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA Automation object.

NO-ERROR
Specifies that any errors that occur in the attempt to release the object are suppressed. After the RELEASE OBJECT statement completes, you can check the ERROR-STATUS system handle for information on any errors that occurred.

For example:

```
RELEASE OBJECT chDataPA NO-ERROR.
```

Visual Studio Syntax

Setting a DataPA object to Nothing in Visual Studio flags it to be released from memory. The syntax for this as follows:

```
DataPA = Nothing
```

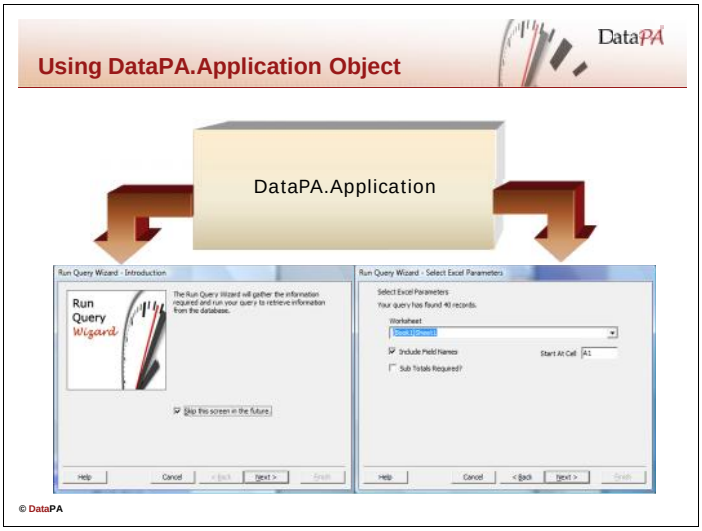
Engine
A variable that references a valid DataPA Application object.

Architect Syntax

The DELETE Object statement destroys a DataPA object and releases the memory it occupied. The syntax for the DELETE Object handle is as follows:

```
IF VALID-OBJECT(ApplicationObj) THEN DELETE OBJECT ApplicationObj.
```

EngineObj
A variable that references a valid DataPA Application object.



Using DataPA.Application Object

The DataPA.Application object provides the core functionality for DataPA to mange all aspects of creating and running queries. The DataPA.Application object does not contain any functionality to export or display data it retrieves. The Application object exposes properties and methods that allow the developer to control all aspects of creating and managing queries. These properties and methods are described below.

The ShowMain Method

The ShowMain method on the application object allows the developer to display the DataPA main window.

AppBuilder Syntax

```
COM-hdl-var:ShowMain(modal, OwnerForm ).
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

modal
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

OwnerForm
A com-handle that references the form that will be deemed the parent to the DataPA main form. You can pass "?" in to this parameter from Progress GUI as the COM-HANDLE of a Progress window is is not readily available.

For example, in Maintenance.w we have the code:

```
chDataPA:ShowMain(1, "?").
```

Visual Studio Syntax

```
Application.ShowMain(modal, OwnerForm)
```

Application
A variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

OwnerForm
A com-handle that references the form that will be deemed the parent to the DataPA main form.
For example:

```
Application.ShowMain(1, Nothing)
```

Architect Syntax

```
Application:ShowMain(expression, OwnerForm)
```

Application
A variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window. Pass "?" if none is to be used.

Owner form
The parent form to be used for the main form of DataPA. Pass "?" if none is to be used.

For example:

```
Application:ShowMain(1,"?").
```

The ShowSetup Method

The ShowSetup method on the application object allows the developer to display the DataPA setup window.

AppBuilder Syntax

```
COM-hdl-var:ShowSetup(expression).
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

For example:

```
chDataPA:ShowSetup(1).
```

Visual Studio Syntax

```
Application.ShowSetup(expression)
```

Application
A variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

For example:

```
Application.ShowSetup(1)
```

Architect Syntax

```
Application:ShowSetup(expression).
```

Application
A variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window. Pass “?” if none is to be used.

Owner form
The parent form to be used for the main form of DataPA. Pass “?” if none is to be used.

For example:

```
Application:ShowSetup(1, "?").
```

The ShowSecurity Method

The ShowSecurity method on the application object allows the developer to display the DataPA security window.

AppBuilder Syntax

```
COM-hdl-var:ShowSecurity(expression).
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

For example, in Maintenance.w we have the code:

```
chDataPA:ShowSecurity(1).
```

Visual Studio Syntax

```
Application.ShowSecurity(expression)
```

Application
A variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

For example:

```
Application.ShowSecurity(1)
```

Architect Syntax

```
Application:ShowSecurity(expression)
```

Application
A variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window. Pass "?" if none is to be used.

Owner form
The parent form to be used for the main form of DataPA. Pass "?" if none is to be used.

For example:

```
Application:ShowSecurity(1,"?").
```

The ShowAbout Method

The ShowAbout method on the application object allows the developer to display the DataPA main about.

AppBuilder Syntax

```
COM-hdl-var:ShowAbout(expression).
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

For example:

```
chDataPA:ShowAbout(1).
```

Visual Studio Syntax

```
Application.ShowAbout(expression)
```

Application
A variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

For example:

```
Application.ShowAbout(1)
```

Architect Syntax

```
Application:ShowAbout(expression)
```

Application
A variable that references a valid DataPA.Application object.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window. Pass "?" if none is to be used.

Owner form
The parent form to be used for the main form of DataPA. Pass "?" if none is to be used.

For example:

```
Application:ShowAbout(1, "?").
```

The ShowExplore Method

The ShowExplore method on the application object allows the developer to display the DataPA subject explorer window.

AppBuilder Syntax

```
COM-hdl-var:ShowExplore(Select, Search[, System] [, Subject]).
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

Select
A logical expression indicating whether or not you wish to show the select panel in the subject explorer window.

Search
A logical expression indicating whether or not you wish to show the search panel in the subject explorer window.

System
An optional string expression that should contain the name of an existing system. If included, the list of subjects will be limited to a particular system.

Subject
An optional string expression to indicate a particular subject with the subject ID. If included the specified subject will be selected initially.

Visual Studio Syntax

```
Application.ShowExplore(Select, Search[, System] [, Subject])
```

Application
A variable that references a valid DataPA.Application object.

Select
A logical expression indicating whether or not you wish to show the select panel in the subject explorer window.

Search
A logical expression indicating whether or not you wish to show the search panel in the subject explorer window.

System
An optional string expression that should contain the name of an existing system. If included, the list of subjects will be limited to a particular system.

Subject
An optional string expression to indicate a particular subject with the subject ID. If included the specified subject will be selected initially.

Architect Syntax

```
Application:ShowExplore(Select, Search, System , Subject).
```

Application
A variable that references a valid DataPA.Application object.

Select
A logical expression indicating whether or not you wish to show the select panel in the subject explorer window.

Search
A logical expression indicating whether or not you wish to show the search panel in the subject explorer window.

System
An optional string expression that should contain the name of an existing system. If included, the list of subjects will be limited to a particular system. Pass “?” if none is to be used.

Subject
An optional string expression to indicate a particular subject with the subject ID. If included the specified subject will be selected initially. Pass “?” if none is to be used.

The ShowQueryWizard Method

The ShowQueryWizard method on the application object allows the developer to display the DataPA Query Wizard screen to create a new query or modify an existing one.

AppBuilder Syntax

```
[Query—COM-hdl-var  =] COM-hdl-var:ShowQueryWizard
                        ([modify-Query—COM-hdl-var], expression).
```

Query-COM-hdl-var
A COM-HANDLE variable that will reference the query object that results from the query wizard if completed successfully.

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

modify-Query—COM-hdl-var
An optional COM-HANDLE variable that references a valid DataPA.Query object. If included, the query wizard will modify the referenced query, otherwise, the query wizard will create a new query.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

Visual Studio Syntax

```
[Query=] Application.ShowQueryWizard (modifyQuery, expression)
```

Query
A variable that will reference the query object that results from the query wizard if completed successfully.

Application
A variable that references a valid DataPA.Application object.

modifyQuery
An optional variable that references a valid DataPA.Query object. If included, the query wizard will modify the referenced query, otherwise, the query wizard will create a new query.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window.

Architect Syntax

```
[Query=] Application:ShowQueryWizard (modifyQuery, expression, owner
form)
```

Query
A variable that will reference the query object that results from the query wizard if completed successfully.

Application
A variable that references a valid DataPA.Application object.

modifyQuery
An optional variable that references a valid DataPA.Query object. If included, the query wizard will modify the referenced query, otherwise, the query wizard will create a new query.

expression
An integer string expression that evaluates to either 1 or 0. 1 indicates that the window should be shown modal, that is the user cannot return to the calling window without shutting the opened window. 0 indicates that the window should be shown no-modal, that is the can return to the calling window without shutting the opened window. Pass "?" if none is to be used.

Owner form
The parent form to be used for the main form of DataPA. Pass "?" if none is to be used.

The LoadQuery Method

The LoadQuery method loads a query from a disk resident file.

AppBuilder Syntax

```
[Query-COM-hdl-var  =] COM-hdl-var:LoadQuery(FileName).
```

Query-COM-hdl-var
A COM-HANDLE variable that will reference the query object that results from the load operation if completed successfully.

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

FileName
A string expression that resolves to a fully qualified operating system file that contains a query.

For example:

```
chQuery = chDataPA:LoadQuery ("C:\OpenEdge\WRK\Customers.qpa").
```

Visual Studio Syntax

```
[Query  =] Application.LoadQuery(FileName)
```

Query
A variable that will reference the query object that results from the load operation if completed successfully.

Application
A variable that references a valid DataPA.Application object.

FileName
A string expression that resolves to a fully qualified operating system file that contains a query.

For example:

```
Query = Application.LoadQuery ("C:\OpenEdge\WRK\Customers.qpa")
```

Architect Syntax

```
[Query  =] Application:LoadQuery(FileName)
```

Query
A variable that will reference the query object that results from the load operation if completed successfully.

Application
A variable that references a valid DataPA.Application object.

FileName
A string expression that resolves to a fully qualified operating system file that contains a query.

For example:

```
Query = Application:LoadQuery (“C:\OpenEdge\WRK\Customers.qpa”).
```

The Query property

The query property on the DataPA.Application object allows the developer to set and retrieve the current loaded query.

AppBuilder Syntax

```
COM-hdl-var:Query.
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

For example, if we wanted to retrieve the handle of the current loaded query, we could use the code:

```
chQuery = chDataPA:Query.
```

If we wanted to set the current loaded query for the DataPA.Application object, we could use the code:

```
chDataPA:Query = chQuery.
```

Visual Studio Syntax

```
Application.Query
```

Application
A variable that references a valid DataPA.Application object.

For example, if we wanted to retrieve the handle of the current loaded query, we could use the code:

```
Query = Application.Query
```

If we wanted to set the current loaded query for the DataPA.Application object, we could use the code:

```
Application.Query = Query
```

Architect Syntax

```
Application:Query
```

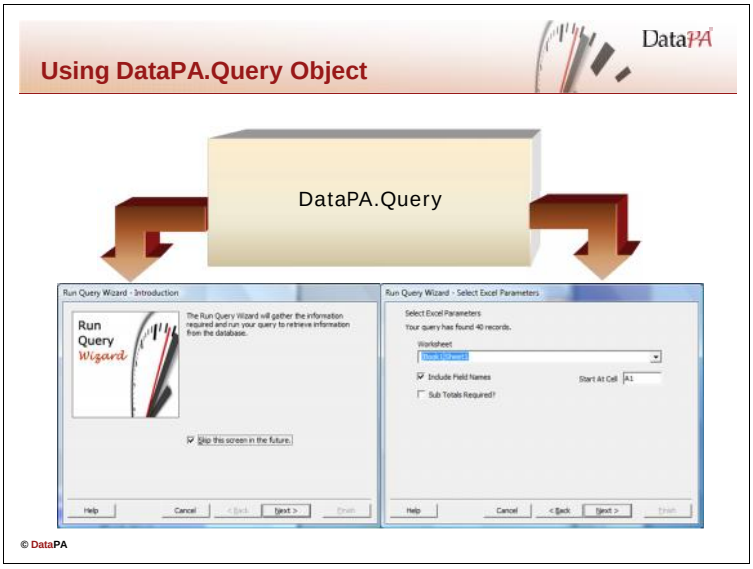
Application
A variable that references a valid DataPA.Application object.

For example, if we wanted to retrieve the handle of the current loaded query, we could use the code:

```
Query = Application:Query().
```

If we wanted to set the current loaded query for the DataPA.Application object, we could use the code:

```
Application:Query = Query().
```



Using DataPA.Query Object

The DataPA.Query object represents a single DataPA query. It provides an interface to allow the developer to maintain the query and run the query. Dealing with parameters on the query object is covered in depth in Lesson 5: Managing Query Parameters. The following section details some of the query parameters and methods that are useful for running queries.

The Query Name Parameter

The Name property on the DataPA.Query object allows the developer to set and retrieve the name of the query.

Appbuilder Syntax

```
COM-hdl-var:Name.
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

For example, if we wanted to retrieve the name of the query, we could use the code:

```
cName = chQuery:Name.
```

If we wanted to set the Name of the DataPA.Query object, we could use the code:

```
chQuery:Name = cName.
```

Visual Studio Syntax

```
Query.Name
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to retrieve the name of the query, we could use the code:

```
sName = chQuery.Name
```

If we wanted to set the Name of the DataPA.Query object, we could use the code:

```
Query.Name = sName
```

Architect Syntax

```
Query:Name
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to retrieve the name of the query, we could use the code:

```
sName = chQuery:Name.
```

If we wanted to set the Name of the DataPA.Query object, we could use the code:

```
Query:Name = sName.
```

The Query Description Parameter

The Description property on the DataPA.Query object allows the developer to set and retrieve the Description of the query.

AppBuilder Syntax

```
COM-hdl-var:Description.
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

For example, if we wanted to retrieve the Description of the query, we could use the code:

```
cDescription = chQuery:Description.
```

If we wanted to set the Description of the DataPA.Query object, we could use the code:

```
chQuery:Description = cDescription.
```

Visual Studio Syntax

```
Query.Description
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to retrieve the Description of the query, we could use the code:

```
sDescription = Query.Description
```

If we wanted to set the Description of the DataPA.Query object, we could use the code:

```
Query.Description = sDescription
```

Architect Syntax

```
Query:Description
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to retrieve the Description of the query, we could use the code:

```
sDescription = Query:Description.
```

If we wanted to set the Description of the DataPA.Query object, we could use the code:

```
Query:Description = sDescription.
```

The Run Query Method

The Run Query method on the DataPA.Query object allows the developer to open the run query wizard to run the query.

AppBuilder Syntax

```
COM-hdl-var:RunQuery.
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

For example, if we wanted to run the query, we could use the code:

```
chQuery:RunQuery.
```

Visual Studio Syntax

```
Query.RunQuery
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to run the query, we could use the code:

```
Query.RunQuery
```

Architect Syntax

```
Query:RunQuery(Object).
```

Query
A variable that references a valid DataPA.Query object.

Object
A used to pass the parent form for the run query wizard in DataPA. This is not supported in Architect but an empty object should be passed as per the example below.

For example, if we wanted to run the query, we could use the code:

```
DEFINE VARIABLE cObject as System.Object NO-UNDO.
Query:RunQuery(cObject).
```

The Query FileName Parameter

The FileName property on the DataPA.Query object allows the developer to set and retrieve the name of the file the query will be saved to or was opened from.

AppBuilder Syntax

```
COM-hdl-var:FileName.
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

For example, if we wanted to set the FileName of the DataPA.Query object, we could use the code:

```
chQuery:FileName = cFileName.
```

Visual Studio Syntax

```
Query.FileName
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to set the FileName of the DataPA.Query object, we could use the code:

```
Query.FileName = sFileName
```

Architect Syntax

```
Query:FileName
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to set the FileName of the DataPA.Query object, we could use the code:

```
Query:FileName = cFileName.
```

The Query Save Method

The Query Save method on the DataPA.Query object allows the developer to save the query to disk.

AppBuilder Syntax

```
COM-hdl-var:Save.
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

For example, if we wanted to save the query to the file returned by chQuery:FileName, we could use the code:

```
chQuery:Save.
```

Visual Studio Syntax

```
Query.Save
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to save the query to the file returned by Query.FileName, we could use the code:

```
Query.Save
```

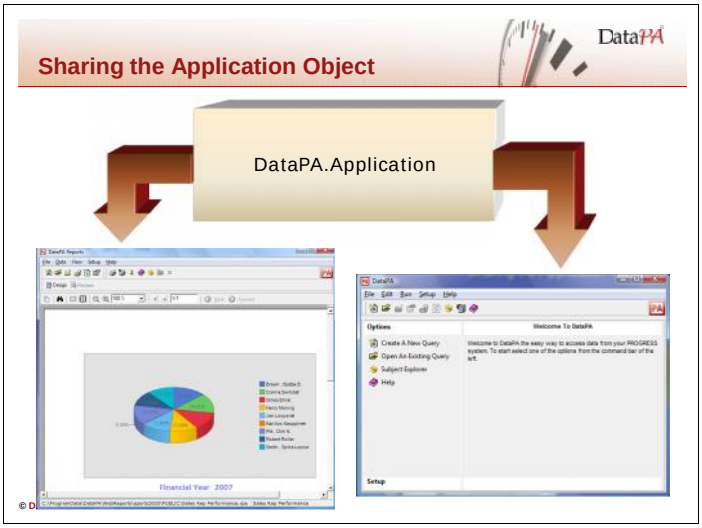
Architect Syntax

```
Query:Save().
```

Query
A variable that references a valid DataPA.Query object.

For example, if we wanted to save the query to the file returned by Query:FileName, we could use the code:

```
Query:Save().
```



Sharing DataPA.Application Object

When the application object is created it runs through an initialisation routine. Depending on your configuration, this routine may involve several AppServer calls and as such can be costly in terms of performance if repeated unnecessarily. As such, it is good practice to make sure that your application creates a single DataPA.Application object, which is used by all procedures that use PAReports or DataPA Automation objects. For the DataPA objects this is simply a case of sharing the handles for the DataPA.Application object.

Assigning a DataPA.Application Object to a PAReports Object

The PAReports Object has a property called ApplicationObject that references the DataPA.Application object used by the PAReports object. This property can be assigned a value BEFORE the Initialise method is called on the PAReports object. If this property has been assigned to a valid DataPA.Application object when the Initialise method is called, the PAReports object will use that DataPA.Application object throughout its life. Otherwise, the PAReports Object will create its own instance of the DataPA.Application object when the Initialise method is called.

AppBuilder Syntax

```
COM-hdl-var:ApplicationObject.
```

COM-hdl-var
A COM-HANDLE variable that references a valid PAReports object.

For example, the following code shows how you can share a DataPA.Application object:

```
DEFINE VARIABLE hApp AS COM-HANDLE      NO-UNDO.

CREATE "DataPA.Application" hApp.
PAReports:ApplicationObject = hApp.
```

Visual Studio Syntax

```
PAReports.ApplicationObject
```

PAReports
A variable that references a valid PAReports object.

For example, the following code shows how you can share a DataPA.Application object:

```
dim hApp AS DataPA.Application

hApp = New DataPA.Application
PAReports:ApplicationObject = hApp
```

Architect Syntax

```
PAReports.ApplicationObject
```

PAReports
A variable that references a valid PAReports object.

For example, the following code shows how you can share a DataPA.Application object:

```
DEFINE VARIABLE hApp AS DataPA.Application NO-UNDO.

hApp = New DataPA.ApplicationClass().
PAReports:ApplicationObject = hApp.
```

Assigning a DataPA.Application to a DashboardDesigner Object

The DashboardDesigner Object has a property called DataPAApplication that references the DataPA.Application object used by the DashboardDesigner object. This property can be assigned a value BEFORE the Initialise method is called on the DashboardDesigner object. If this property has been assigned to a valid DataPA.Application object when the Initialise method is called, the DashboardDesigner object will use that DataPA.Application object throughout its life. Otherwise, the DashboardDesigner object will create its own instance of the DataPA.Application object when the Initialise method is called.

AppBuilder Syntax

```
COM-hdl-var:DataPAApplication.
```

COM-hdl-var
A COM-HANDLE variable that references a valid DashboardDesigner object.

For example, the following code shows how you can share a DataPA.Application object:

```
DEFINE VARIABLE hApp AS COM-HANDLE      NO-UNDO.

CREATE “DataPA.Application” hApp.
DashboardDesigner:DataPAApplication = hApp.
```

Visual Studio Syntax

```
DashboardDesigner.DataPAApplication
```

DashboardDesigner
A variable that references a valid DashboardDesigner object.

For example, the following code shows how you can share a DataPA.Application object:

```
dim hApp AS DataPA.Application

hApp = New DataPA.Application
DashboardDesigner:DataPAApplication = hApp
```

Architect Syntax

```
DashboardDesigner.DataPAApplication
```

DashboardDesigner
A variable that references a valid DashboardDesigner object.

For example, the following code shows how you can share a DataPA.Application object:

```
DEFINE VARIABLE hApp AS DataPA.Application NO-UNDO.

hApp = New DataPA.ApplicationClass().
DashboardDesigner:DataPAApplication = hApp.
```

Managing Query Parameters

We have already seen in the previous lessons how we can use the DataPA.Query object to open and run queries. This lesson develops those techniques so we can manage query parameters and run queries without DataPA prompting the user for input. These techniques are applicable to DataPA queries run from PAReports objects and DataPA Application Objects.

Learning Objectives

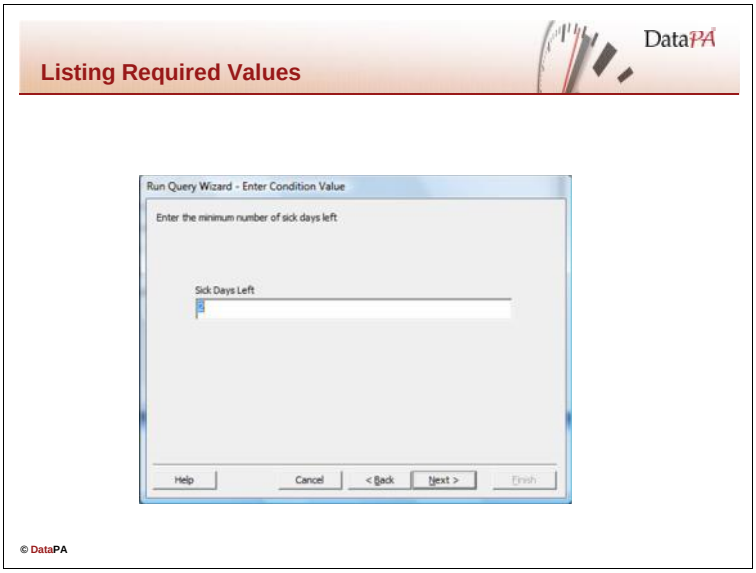
When you complete this lesson you should be able to:

- List the input parameters that a query requires to run
- Programmatically set input parameters values
- Programmatically add query parameters
- Use the query parameters to hide sections of the run query wizard

Prerequisites

Before you begin this lesson you should be able to:

- Create a GUI application using Progress
- Use methods and properties on objects
- Use the DataPA Application objects and PAReports controls to run queries



Listing Required Values

Required values occur when conditions are created against the subject or query, that have been designated as *Required Values* in either the subject or query wizard. In normal operation, DataPA prompts the user for values to satisfy these required values in the query wizard at runtime. However, when imbedding DataPA, we may often wish the application to set these required values, for instance if we are designing an invoice viewer, we may want the Application to select the Invoice to be displayed.

The Required Values Collection

The Query object has a property called RequiredFields, that returns a collection of Field objects, one for each value that is required to satisfy the query parameters.

AppBuilder Syntax

```
COM-hdl-var:RequiredFields
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

The RequiredFields property returns a COM-Handle that represents the RequiredFields collection.

The RequiredFields collection has one property and one method that allow us to retrieve information about the required fields. First is the Count property that retrieves the number of required values for the query. The syntax for the Count property is as follows:

```
COM-hdl-var:Count
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query.RequiredFields object.

To retrieve individual required fields, the required fields property has an Item method. The Item method has the following syntax:

```
COM-hdl-var:item(expression)
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query.RequiredFields object.

expression
An integer or string expression that resolves to either the position of the required field in the list of required fields, or the name of the required field.

The Item method returns a COM-handle that represents a DataPA.Fields object.

The table below summarised the properties of the fields object:

Property	Type	Description
Name	Character	The name of the required field
Label	Character	The label used when prompting for the required field
Description	Character	The description used when prompting for the required field
DataType	Character	The Progress data type of the required field
Mandatory	Logical	Indicates whether the required field is mandatory or not
Format	Character	The Progress format of the required field

The following example shows a message for each field in a query with the name, label and description.

```
DEFINE VARIABLE iNumParams AS INTEGER    NO-UNDO.
DEFINE VARIABLE ix         AS INTEGER    NO-UNDO.

ASSIGN iNumParams = chQuery:RequiredFields:COUNT.

IF iNumParams > 0 THEN REPEAT WITH ix = 1 TO iNumParams:
   MESSAGE chQuery:RequiredFields:ITEM(ix):NAME SKIP
           chQuery:RequiredFields:ITEM(ix):LABEL SKIP
           chQuery:RequiredFields:ITEM(ix):DESCRIPTION
     VIEW-AS ALERT-BOX INFO BUTTONS OK.
END.
```

Visual Studio Syntax

```
Query.RequiredFields
```

Query
A variable that references a valid DataPA.Query object.

The RequiredFields property returns a handle that represents the RequiredFields collection.

The RequiredFields collection has one property and one method that allow us to retrieve information about the required fields. First is the Count property that retrieves the number of required values for the query. The syntax for the Count property is as follows:

```
RequiredFields.Count
```

RequiredFields
A variable that references a valid DataPA.Query.RequiredFields object.

To retrieve individual required fields, the required fields property has an Item method. The Item method has the following syntax:

```
RequiredFields.item(expression)
```

RequiredFields
A variable that references a valid DataPA.Query.RequiredFields object.

expression
An integer or string expression that resolves to either the position of the required field in the list of required fields, or the name of the required field.

The Item method returns a handle that represents a DataPA.Fields object.

The table below summarised the properties of the fields object:

Property	Type	Description
Name	Character	The name of the required field
Label	Character	The label used when prompting for the required field
Description	Character	The description used when prompting for the required field
DataType	Character	The Progress data type of the required field
Mandatory	Logical	Indicates whether the required field is mandatory or not
Format	Character	The Progress format of the required field

The following example shows a message for each field in a query with the name, label and description.

```
dim iNumParams as integer
dim ix         as integer

iNumParams = Query.RequiredFields.COUNT

IF iNumParams > 0 then
  For ix = 1 to iNumParams
    msgbox Query.RequiredFields.ITEM(ix).NAME
           Query.RequiredFields.ITEM(ix).LABEL
           Query.RequiredFields.ITEM(ix).DESCRIPTION
  Next ix
end if
```

Architect Syntax

```
Query:RequiredFields
```

Query
A variable that references a valid DataPA.Query object.

The RequiredFields property returns a handle that represents the RequiredFields collection.

The RequiredFields collection has one property and one method that allow us to retrieve information about the required fields. First is the Count property that retrieves the number of required values for the query. The syntax for the Count property is as follows:

```
RequiredFields:Count
```

RequiredFields
A variable that references a valid DataPA.Query.RequiredFields object.

To retrieve individual required fields, the required fields property has an Item method. The Item method has the following syntax:

```
RequiredFields:item(expression)
```

RequiredFields
A variable that references a valid DataPA.Query.RequiredFields object.

expression
An integer or string expression that resolves to either the position of the required field in the list of required fields, or the name of the required field.

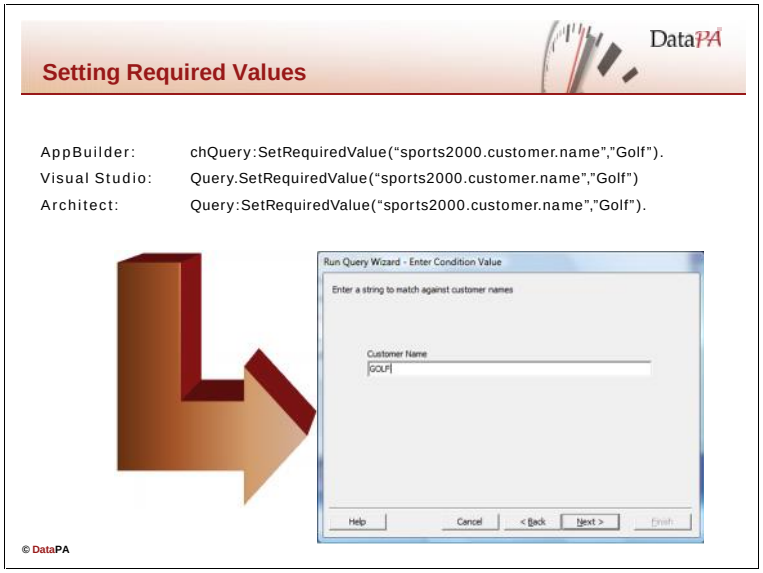
The Item method returns a handle that represents a DataPA.Fields object.

The table below summarised the properties of the fields object:

Property	Type	Description
Name	Character	The name of the required field
Label	Character	The label used when prompting for the required field
Description	Character	The description used when prompting for the required field
DataType	Character	The Progress data type of the required field
Mandatory	Logical	Indicates whether the required field is mandatory or not
Format	Character	The Progress format of the required field

The following example shows a message for each field in a query with the name, label and description.

```
DEFINE VARIABLE iNumParams AS INTEGER      NO-UNDO.
DEFINE VARIABLE ix         AS INTEGER      NO-UNDO.

ASSIGN iNumParams = Query:RequiredFields:COUNT().

IF iNumParams > 0 THEN REPEAT WITH ix = 1 TO iNumParams:
   MESSAGE Query:RequiredFields:ITEM(ix):NAME SKIP
           Query:RequiredFields:ITEM(ix):LABEL SKIP
           Query:RequiredFields:ITEM(ix):DESCRIPTION
     VIEW-AS ALERT-BOX INFO BUTTONS OK.
END.
```



Setting Required Valies

Often we will want to programmatically set the value of required fields for a query rather than letting the user enter the values in the query wizard.

The SetRequiredField Method

The SetRequiredField method allows us to programmatically set a value of a required field. If a required field value is set programmatically BEFORE we call the RunQuery method, the query wizard will not prompt the user for that value.

AppBuilder Syntax

```
COM-hdl-var:SetRequiredValue(FieldName, value)
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

FieldName
A character expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

Value
An expression that represents the value you wish to assign to the field. The value should be the data type specified by the DataType property of the Field object.

For example, the following code sets the value of the sports2000.customer.name required field to *Golf*.

```
chQuery:chQuery:SetRequiredValue("sports2000.customer.name","Golf")
```

Visual Studio Syntax

```
Query.SetRequiredValue(FieldName, value)
```

Query
A variable that references a valid DataPA.Query object.

FieldName
A character expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

Value
An expression that represents the value you wish to assign to the field. The value should be the data type specified by the DataType property of the Field object.

For example, the following code sets the value of the sports2000.customer.name required field to *Golf*.

```
Query.SetRequiredValue("sports2000.customer.name","Golf")
```

Architect Syntax

```
Query:SetRequiredValue(FieldName, value)
```

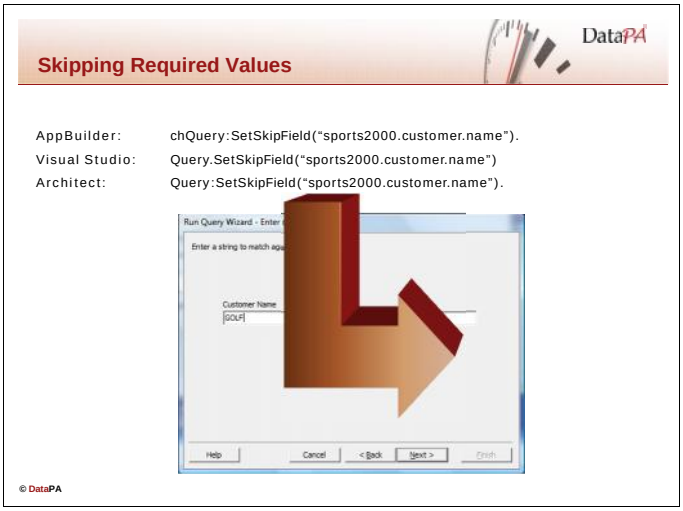
Query
A variable that references a valid DataPA.Query object.

FieldName
A character expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

Value
An expression that represents the value you wish to assign to the field. The value should be the data type specified by the DataType property of the Field object.

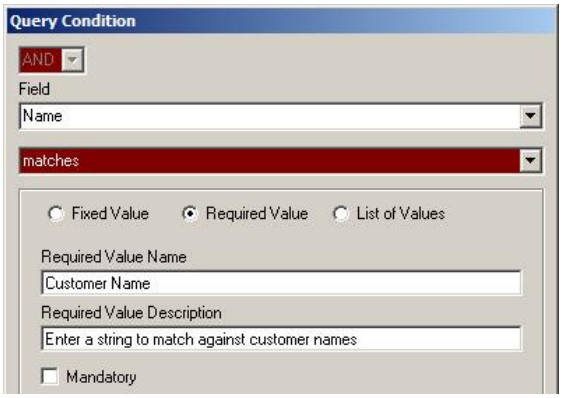
For example, the following code sets the value of the sports2000.customer.name required field to *Golf*.

```
Query:SetRequiredValue("sports2000.customer.name","Golf").
```



Skipping Required Values

Required values can sometimes be optional. If the administrator did not check the mandatory option when creating the required field, the user is given the option to skip the required field.



The SetSkipField method

You can programmatically skip a required field using the SetSkipField method. If the SetSkipField method is called on a non-mandatory required field BEFORE the RunQuery method is called, the required field will not be prompted for by the run query wizard.

AppBuilder Syntax

```
COM-hdl-var:SetSkipValue(FieldName)
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

FieldName
A character expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

For example, the following code tells the Run Query Wizard to skip the required field sports2000.customer.name.

```
chQuery:chQuery:SetSkipValue("sports2000.customer.name")
```

Visual Studio Syntax

```
Query.SetSkipValue(FieldName)
```

Query
A variable that references a valid DataPA.Query object.

FieldName
A character expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

For example, the following code tells the Run Query Wizard to skip the required field sports2000.customer.name.

```
Query.SetSkipValue("sports2000.customer.name")
```

Architect Syntax

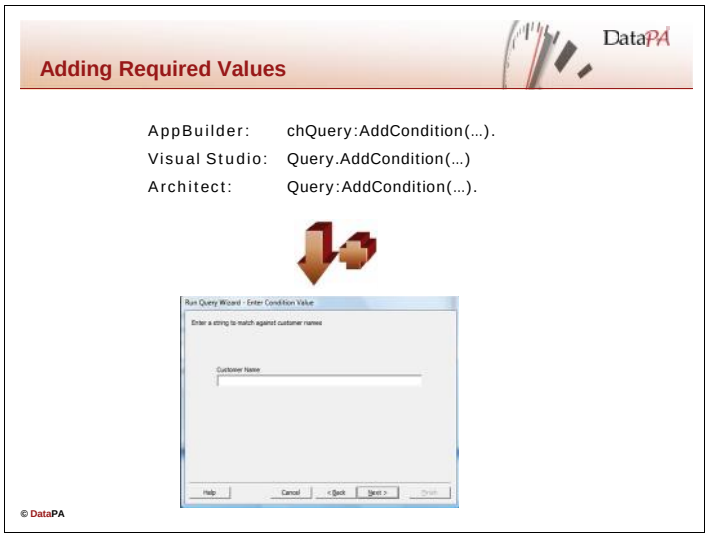
```
Query:SetSkipValue(FieldName)
```

Query
A variable that references a valid DataPA.Query object.

FieldName
A character expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

For example, the following code tells the Run Query Wizard to skip the required field sports2000.customer.name.

```
Query:SetSkipValue("sports2000.customer.name").
```



Adding Required Values

Sometimes it may be necessary to programmatically add conditions to a query rather than through the query wizard.

The AddCondition Method

The AddCondition Method allows the developer to programmatically add a condition to the query.

AppBuilder Syntax

```
COM-hdl-var:AddCondition(LogicalOperation,
                         FieldName,
                         Operator,
                         RequiredField,
                         Expression,
                         Label,
                         Description,
                         Mandatory,
                         OpenBrackets,
                         CloseBrackets).
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

LogicalOperation
A character expression that resolves to either “AND” or “OR”. Represents the logical operator that will be applied to the condition when added to the *FOR EACH* statement.

FieldName
A character expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

Operator
A character expression that resolves to valid Progress operator for the field.

RequiredField
A logical expression. If True, the condition will be created as a required field condition, otherwise the condition will be created as a fixed value condition.
Expression

A character expression that resolves to a valid expression for a fixed value condition.

Label
A character expression that will be used as the label for a required field condition.

Description
A character expression that will be used as the description for a required field condition.

Mandatory
A logical expression that will determine whether a required field condition is mandatory or not.

OpenBrackets
An integer expression that equates to the number of opening brackets that should precede the condition in the *FOR EACH* statement.

CloseBrackets
An integer expression that equates to the number of closing brackets that should follow the condition in the *FOR EACH* statement.

For example, the following code adds a non mandatory required field value to the query:

```
chQuery:AddCondition("AND",
                     "sports2000.customer.custnum",
                     "matches",
                     TRUE,
                     "",
                     "Customer Name",
                     "Enter a string to match against customer name",
                     FALSE,
                     0,
                     0).
```

Visual Studio Syntax

```
Query.AddCondition(LogicalOperation,
                           FieldName,
                           Operator,
                           RequiredField,
                           Expression,
                           Label,
                           Description,
                           Mandatory,
                           OpenBrackets,
                           CloseBrackets)
```

Query
A variable that references a valid DataPA.Query object.

LogicalOperation
A string expression that resolves to either "AND" or "OR". Represents the logical operator that will be applied to the condition when added to the *FOR EACH* statement.

FieldName
A string expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

Operator
A string expression that resolves to valid Progress operator for the field.

RequiredField
A boolean expression. If True, the condition will be created as a required field condition, otherwise the condition will be created as a fixed value condition.

Expression
A string expression that resolves to a valid expression for a fixed value condition.

Label
A string expression that will be used as the label for a required field condition.

Description
A string expression that will be used as the description for a required field condition.

Mandatory
A boolean expression that will determine whether a required field condition is mandatory or not.

OpenBrackets
An integer expression that equates to the number of opening brackets that should precede the condition in the *FOR EACH* statement.

CloseBrackets
An integer expression that equates to the number of closing brackets that should follow the condition in the *FOR EACH* statement.

For example, the following code adds a non mandatory required field value to the query:

```
    Query.AddCondition("AND",
                       "sports2000.customer.custnum",
                       "matches",
                       TRUE,
                       "",
                       "Customer Name",
                       "Enter a string to match against customer name",
                       FALSE,
                       0,
                       0)
```

Architect Syntax

```
    Query:AddCondition(LogicalOperation,
                             FieldName,
                             Operator,
                             RequiredField,
                             Expression,
                             Label,
                             Description,
                             Mandatory,
                             OpenBrackets,
                             CloseBrackets)
```

Query
A variable that references a valid DataPA.Query object.

LogicalOperation
A character expression that resolves to either "AND" or "OR". Represents the logical operator that will be applied to the condition when added to the *FOR EACH* statement.

FieldName

A character expression that resolves to a field name. The field name should match exactly the name value returned from the Field object.

Operator
A character expression that resolves to valid Progress operator for the field.

RequiredField
A logical expression. If True, the condition will be created as a required field condition, otherwise the condition will be created as a fixed value condition.
Expression
A character expression that resolves to a valid expression for a fixed value condition.

Label
A character expression that will be used as the label for a required field condition.

Description
A character expression that will be used as the description for a required field condition.

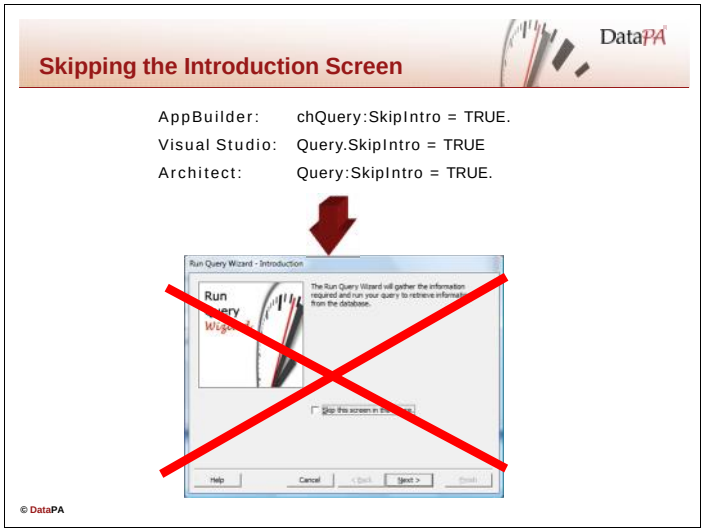
Mandatory
A logical expression that will determine whether a required field condition is mandatory or not.

OpenBrackets
An integer expression that equates to the number of opening brackets that should precede the condition in the *FOR EACH* statement.

CloseBrackets
An integer expression that equates to the number of closing brackets that should follow the condition in the *FOR EACH* statement.

For example, the following code adds a non mandatory required field value to the query:

```
Query:AddCondition("AND",
                   "sports2000.customer.custnum",
                   "matches",
                   TRUE,
                   "",
                   "Customer Name",
                   "Enter a string to match against customer name",
                   FALSE,
                   0,
                   0).
```



Skipping the Introduction Screen

Once all the required parameters have been set, the only user-interaction required in the run-query wizard before the query runs is to skip past the introduction screens. If the run query process is being handled programmatically, you may wish to remove all user-interaction in the wizard so the process is automatic. The first step to do this is to skip the Introduction screen.

The SkipIntro property

The SkipIntro property of the query object specifies whether or not DataPA skips the introduction screen when the query is run.

AppBuilder Syntax

```
COM-hdl-var:SkipIntro = expression
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

expression
A logical expression that resolves to either TRUE or FALSE. If TRUE, the run query wizard will skip the introduction screen, otherwise the run query wizard will show the introduction screen.

For example, the following code tells DataPA to skip the run query wizards introduction screen when a specific query is run:

```
chQuery:SkipIntro = TRUE.
```

Visual Studio Syntax

```
Query.SkipIntro = expression
```

Query
A variable that references a valid DataPA.Query object.

expression
A boolean expression that resolves to either TRUE or FALSE. If TRUE, the run query wizard will skip the introduction screen, otherwise the run query wizard will show the introduction screen.

For example, the following code tells DataPA to skip the run query wizards introduction screen when a specific query is run:

```
Query.SkipIntro = TRUE
```

Architect Syntax

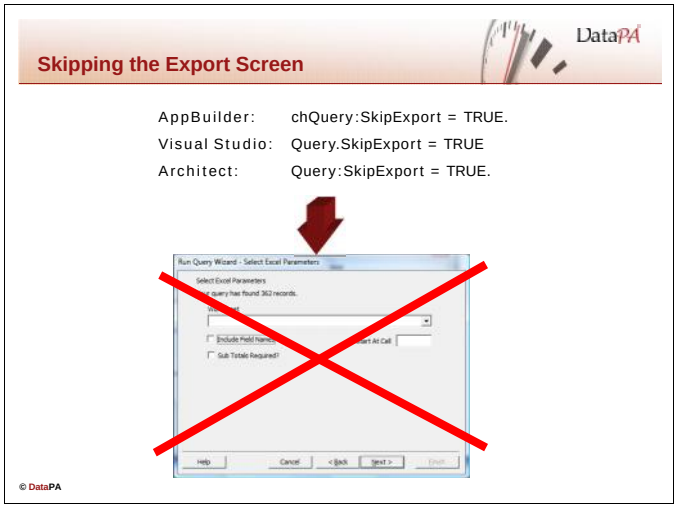
```
Query:SkipIntro = expression
```

Query
A variable that references a valid DataPA.Query object.

expression
A logical expression that resolves to either TRUE or FALSE. If TRUE, the run query wizard will skip the introduction screen, otherwise the run query wizard will show the introduction screen.

For example, the following code tells DataPA to skip the run query wizards introduction screen when a specific query is run:

```
Query:SkipIntro = TRUE.
```



Skipping the Export Screen

Once all the required parameters have been set, the only user-interaction required at the end of the run query wizard is the export information screens. If the run query process is being handled programmatically, you may wish to remove all user-interaction in the wizard so the process is automatic. The last step to do this is to skip the Export screen.

The SkipExport property

The SkipExport property of the query object specifies whether or not DataPA skips the export screen when the query is run.

AppBuilder Syntax

```
COM-hdl-var:SkipExport = expression
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query object.

expression
A logical expression that resolves to either TRUE or FALSE. If TRUE, the run query wizard will skip the export screen, otherwise the run query wizard will show the export screen.

For example, the following code tells DataPA to skip the run query wizards export screen when a specific query is run:

```
chQuery:SkipExport = TRUE.
```

Visual Studio Syntax

```
Query.SkipExport = expression
```

Query
A variable that references a valid DataPA.Query object.

expression
A boolean expression that resolves to either TRUE or FALSE. If TRUE, the run query wizard will skip the export screen, otherwise the run query wizard will show the export screen.

For example, the following code tells DataPA to skip the run query wizards export screen when a specific query is run:

```
Query.SkipExport = TRUE
```

Architect Syntax

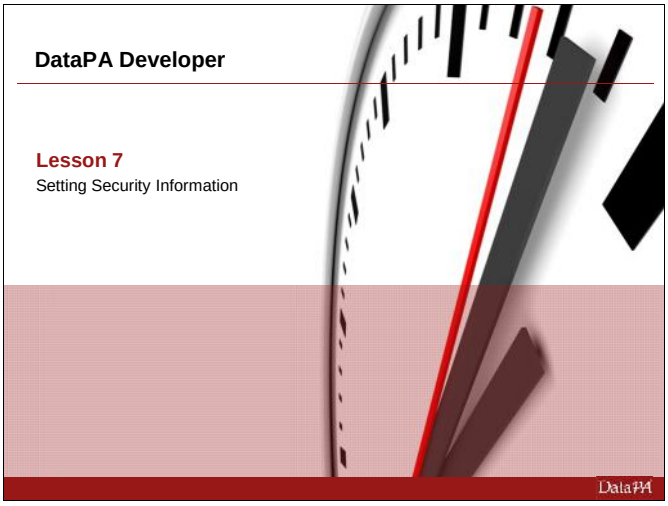
```
Query:SkipExport = expression.
```

Query
A variable that references a valid DataPA.Query object.

expression
A logical expression that resolves to either TRUE or FALSE. If TRUE, the run query wizard will skip the export screen, otherwise the run query wizard will show the export screen.

For example, the following code tells DataPA to skip the run query wizards export screen when a specific query is run:

```
Query:SkipExport = TRUE.
```



Setting Security Information

This lesson will teach the techniques required to automatically manage the DataPA installation and its security automatically from the client application. The techniques use a combination of DataPA Automation Object properties and methods to manage these tasks.

Learning Objectives

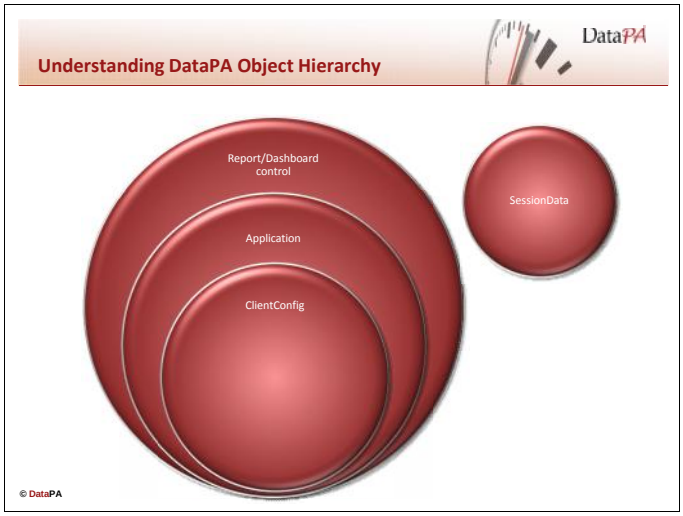
When you complete this lesson you should be able to:

- Understand the hierarchy of the DataPA client objects
- Use the code to programmatically set the DataLocation for DataPA
- Use the code to set the username and password
- Use code to set whether the user should be prompted for the username/password

Prerequisites

Before you begin this lesson you should be able to:

- Create a GUI application in the development tool you have chosen
- Use methods and properties on objects
- Understand how to set DataPA to load data files from the server using the DataLocation



The ClientConfig Object

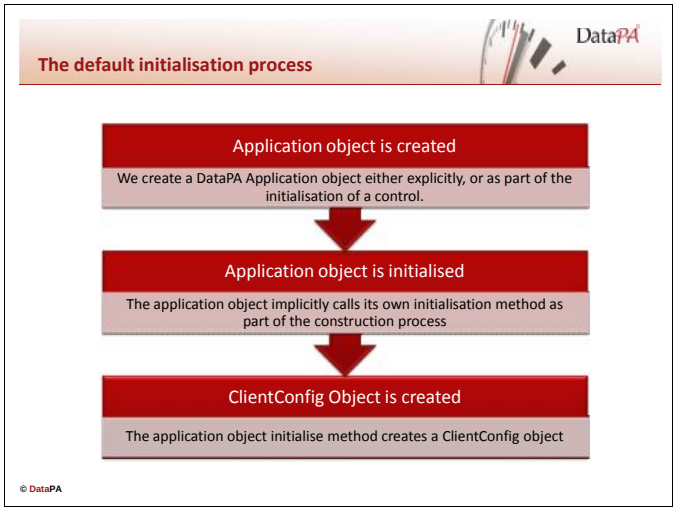
The ClientConfig object provides the core client configuration behaviour for any graphical DataPA client application. It is this object that provides the functionality to open and save the setup from the data location and the core security features of the DataPA client applications.

The Application Object

The DataPA Application object is described in detail in a previous lesson. It provides the functionality to manage and run queries. Every instance of a DataPA Application object must have a ClientConfig object, which by default it will create on initialisation. We will see later in this lesson how we can override this default behaviour to provide a ClientConfig object with that overrides the default behaviour.

The SessionData Object

The SessionData object is a general purpose object that allows us to set session wide variables that can be read by any of the DataPA objects. We will see later in this lesson how we can use the SessionData object to set a session only data location and delay the initialisation of the Application object.

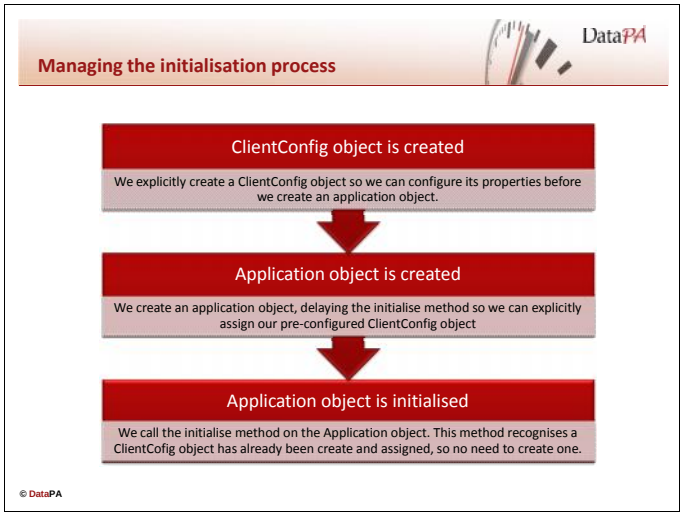


The default initialisation process

If we explicitly or implicitly (as part of the initialisation routine of one of the DataPA controls) create a DataPA Application object, the following steps occur;

1. The DataPA Application object is created.
2. The object calls an internal initialisation procedure as part of its own constructor process.
3. The initialise method of the Application object creates and initialises a ClientConfig object.

This process is efficient in terms of the amount of code we need to write, but does not give us any control of the initialisation process. In many instances we may wish to check or change some settings, such as applying username and password details so the login screen does not show. We need to do this BEFORE the ClientConfig object is initialised in order to change the behaviour of the initialisation process.



Managing the initialisation process

We can modify the default initialisation process by doing the following (the steps are described in more detail later in this lesson);

1. Explicitly create the ClientConfig object
2. Modify any properties we wish to modify before the object is initialised.
3. Create a SessionData object and set the DelayInitialise property to delay the Application object initialise method.
4. Create the application object.
5. Assign our configured ClientConfig object to the Application object.
6. Explicitly call the Application object initialise method.

Creating a ClientConfig object

To create a ClientConfig object, use the following syntax;

AppBuilder Syntax

```
CREATE "DataPAClientConfig.ClientConfig" chClientConfig.
```

Visual Studio Syntax

```
ClientConfig = New DataPAClientConfig.ClientConfig
```

Architect Syntax

```
ClientConfig = NEW DataPAClientConfig.ClientConfig().
```

Creating a SessionData object

We will use the SessionData object to delay the initialisation of the Application object. To create a SessionData object, use the following syntax;

AppBuilder Syntax

```
CREATE "DataPA.SessionData" chSessionData.
```

Visual Studio Syntax

```
SessionData = New DataPA.SessionData
```

Architect Syntax

```
SessionData = NEW DataPA.SessionDataClass().
```

Delaying the Application object Initialise method

We need to delay the call to the initialise method when we create the DataPA Application object so we can assign our managed ClientConfig object rather than allowing the application object to create its own. To do this we must set the DelayedRender value on a SessionData object to True before we create the Application object. The syntax to set a value on the SessionData object is as follows;

AppBuilder Syntax

```
chSessionData:value("DelayInitialise") = True.
```

Visual Studio Syntax

```
SessionData.value("DelayInitialise") = True
```

Architect Syntax

```
SessionData:SetComValue("DelayInitialise", "True").
```

Calling the Application object initialise method

Once we have created our Application object, we must assign our ClientConfig object and then call the initialise event. The syntax for the full example should be;

AppBuilder Syntax

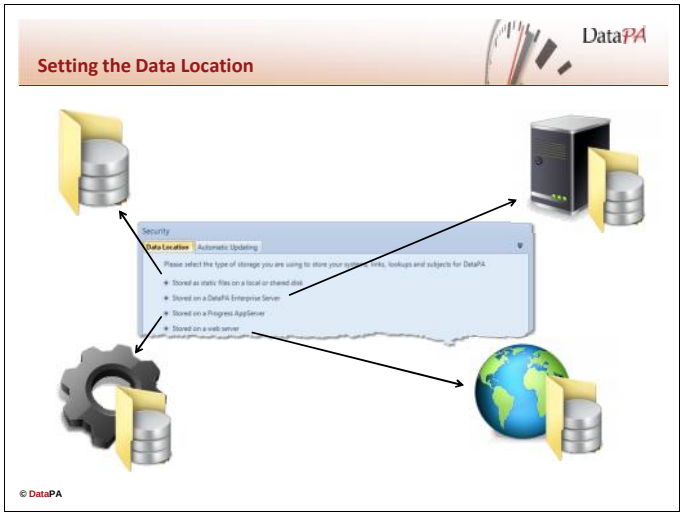
```
DEFINE VARIABLE chClientConfig AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE chSessionData AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE chDataPA AS COM-HANDLE NO-UNDO.

CREATE "DataPA.SessionData" chSessionData.
chSessionData:value("DelayInitialise") = True.
CREATE "DataPAClientConfig.ClientConfig" chClientConfig.

/* Set any properties here */
CREATE "DataPA.Application" chDataPA.
chDataPA:ClientConfig = chClientConfig.
chDataPA:Initialise().
```

Visual Studio Syntax

```
Dim SessionData As New DataPA.SessionData

SessionData.value("DelayInitialise") = True
Dim ClientConfig As New DataPAClientConfig.ClientConfig

' Set any properties here
Dim DataPA As New DataPA.Application
DataPA.ClientConfig = ClientConfig
DataPA.Initialise()
```

Architect Syntax

```
DEFINE VARIABLE clientConfig AS DataPAClientConfig.ClientConfig.
DEFINE VARIABLE SessionData AS DataPA.SessionData.
DEFINE VARIABLE DataPAApplication AS DataPA.Application.

SessionData = NEW DataPA.SessionDataClass().
SessionData:SetComValue("DelayInitialise", "True").
clientConfig = NEW DataPAClientConfig.ClientConfig().

/* Set any properties here */
DataPAApplication = NEW DataPA.ApplicationClass().
DataPAApplication:ClientConfig = clientConfig.
DataPAApplication:Initialise().
```



Setting the Data Location

The Data Location is used to locate and open the data files that contain the configuration for the analytics engine (the definitions of your systems, links, lookups and subjects). It is common to set the Data Location to a server, so the data files can be located centrally and shared rather than locally on each client machine. To ensure the Data Location is not hard coded and can be changed to suit the requirements of different implementations, it is preferable to set the Data Location programmatically. This can be done in two different ways depending on your requirements, either for all sessions or just for the current session. Both of these methods are described in detail later in this lesson.

Data Location syntax

The data location property is a string, the syntax of which determines whether the client will attempt to read the data files from a file location, an enterprise server, an AppServer or a web server. The syntax of the data location property for each possible location type is detailed below;

File Location

The file location syntax requires a fully qualified path to the directory that contains the data files. This includes a trailing slash.

Syntax: [Fully Qualified Path]
Example: C:\ProgramData\DataPA\

Enterprise Server

The Enterprise Server syntax requires the server address followed by the enterprise service details.

Syntax: [Server Address]:80/DataPAServiceModel/EnterpriseService
Example: localhost:80/DataPAServiceModel/EnterpriseService

AppServer

The AppServer syntax requires the AppServer URL. This can be copied directly from the string built by the security string.

Syntax: AppServer[DC]://<HOST>[:<PORT NUMBER>]/<APPSERVICE>
[?[username=<USERNAME>][&password=<PASSWORD>][&prompt=true]
[&appserverinfo=<APPSERVERINFO]]
Example: AppServer://localhost/sports2000?prompt=true

Web Server
The web server syntax simply requires the fully qualified URL.

Syntax: [Fully qualified URL]
Example: http://www.datapa.com

Setting the Data Location for all Sessions

The ClientConfig object has a Data Location property that is directly associated with the data location configured through the security screen. This means if the data location is changed using the ClientConfig property, the change will affect all subsequent DataPA client applications opened on the same machine.

AppBuilder Syntax

```
COM-hdl-var:DataLocation
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA ClientConfig object.

For example, if we wanted to set default location we could use the code:

```
DEFINE VARIABLE chClientConfig AS COM-HANDLE NO-UNDO.
CREATE "DataPAClientConfig.ClientConfig" chClientConfig.
chClientConfig:DataLocation = "C:\ProgramData\DataPA\".
RELEASE OBJECT chClientConfig.
```

Visual Studio Syntax

```
ClientConfig.DataLocation
```

AdminObj
A variable that references a valid DataPA ClientConfig object.

For example, if we wanted to set our DataLocation to the default location we could use the code:

```
Dim ClientConfig As New DataPAClientConfig.ClientConfig
ClientConfig.DataLocation = "C:\ProgramData\DataPA\"
```

Architect Syntax

```
ClientConfig:DataLocation
```

ClientConfig
A variable that references a valid DataPA ClientConfig object.

For example, if we wanted to set our DataLocation to the default location we could use the code:

```
DEFINE VARIABLE clientConfig AS DataPAClientConfig.ClientConfig.
clientConfig = NEW DataPAClientConfig.ClientConfig().
clientConfig:DataLocation = “C:\ProgramData\DataPA\”.
```

Setting the Data Location for a single session

You may choose to have two completely separate configurations for the analytics engine, depending on whether DataPA OpenAnalytics is being used as an embedded technology, or stand alone. This is particularly useful when you wish to allow customers to create their own subjects, but prevent them from modifying subjects used in the application.

To support this, we can set a data location property on the session data object. Any DataPA client objects created in the same session (process) during the lifetime of this object will use the given data location. The examples below give the syntax using the delayed initialization discussed earlier in this chapter.

Syntax

AppBuilder Syntax

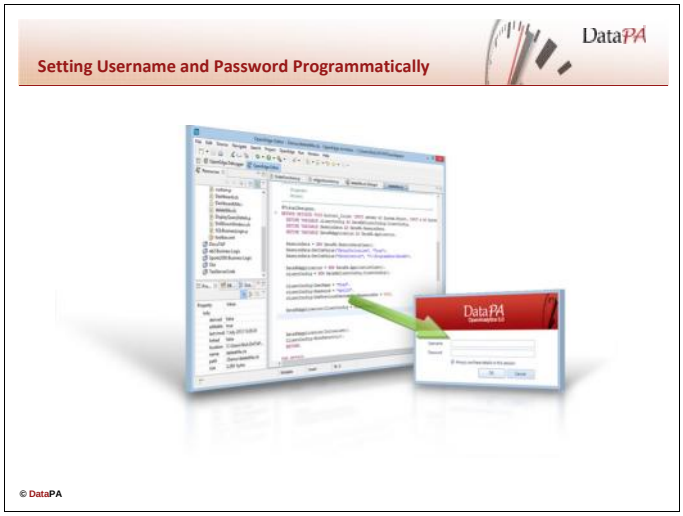
```
DEFINE VARIABLE chClientConfig AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE chSessionData AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE chDataPA AS COM-HANDLE NO-UNDO.

CREATE "DataPA.SessionData" chSessionData.
chSessionData:value("DelayInitialise") = True.
chSessionData:value("DataLocation") = "C:\ProgramData\DataPA".

CREATE "DataPAClientConfig.ClientConfig" chClientConfig.
CREATE "DataPA.Application" chDataPA.
chDataPA:ClientConfig = chClientConfig.
chDataPA:Initialise().
```

Visual Studio Syntax

```
Dim SessionData As New DataPA.SessionData

SessionData.value("DelayInitialise") = True
SessionData.value("DataLocation") = "C:\ProgramData\DataPA"
Dim ClientConfig As New DataPAClientConfig.ClientConfig

Dim DataPA As New DataPA.Application
DataPA.ClientConfig = ClientConfig
DataPA.Initialise()
```

Architect Syntax

```
DEFINE VARIABLE clientConfig AS DataPAClientConfig.ClientConfig.
DEFINE VARIABLE SessionData AS DataPA.SessionData.
DEFINE VARIABLE DataPAApplication AS DataPA.Application.

SessionData = NEW DataPA.SessionDataClass().
SessionData:SetComValue("DelayInitialise", "True").
SessionData:SetComValue("DataLocation", "C:\ProgramData\DataPA").
clientConfig = NEW DataPAClientConfig.ClientConfig().

DataPAApplication = NEW DataPA.ApplicationClass().
DataPAApplication:ClientConfig = clientConfig.
DataPAApplication:Initialise().
```



Setting Username and Password Programmatically

More often than not, any connection to an AppServer or DataPA Enterprise server will require authentication. It is often the case that the username and password required is the same as that provided by the user when logging in to the business application. To save the user entering their username and password multiple times, it is good practice to apply the username and password programmatically.

Setting Username and Password globally

There are two different situations where DataPA connects to a server and can potentially require a username and password. If the data location is set to a DataPA Enterprise server, or AppServer and the server requires authentication, a username and password is required when the ClientConfig object is initialised. If a connection defined within a system requires a username and password, DataPA OpenAnalytics will prompt for a username and password when a connection is required.

If there is only a single server that needs to be authenticated against, or the username and password for all servers is consistent, we can use the ClientConfig object to set the username and password and remember it. This method behaves exactly the same as providing the username and password and checking the *Always use these details in this session* check box. That is, every time DataPA would prompt for a username and password, it will try the saved details and only show the login screen if these details fail.

The examples below shows how to use the username, password and UsePreviousUsernameAndPasswordSet properties with the delayed initialisation method described earlier in this lesson.

Syntax

AppBuilder Syntax

```
DEFINE VARIABLE chClientConfig AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE chSessionData AS COM-HANDLE NO-UNDO.
DEFINE VARIABLE chDataPA AS COM-HANDLE NO-UNDO.

CREATE "DataPA.SessionData" chSessionData.
chSessionData:value("DelayInitialise") = True.
CREATE "DataPAClientConfig.ClientConfig" chClientConfig.

chClientConfig:UserName = "Guest".
chClientConfig:Password = "openanalytics".
chClientConfig:UsePreviousUsernameAndPasswordSet = True.

CREATE "DataPA.Application" chDataPA.
chDataPA:ClientConfig = chClientConfig.
chDataPA:Initialise().
```

Visual Studio Syntax

```
Dim SessionData As New DataPA.SessionData

SessionData.value("DelayInitialise") = True
Dim ClientConfig As New DataPAClientConfig.ClientConfig

ClientConfig.UserName = "Guest"
ClientConfig.Password = "openanalytics"
ClientConfig.UsePreviousUsernameAndPasswordSet = True

Dim DataPA As New DataPA.Application
DataPA.ClientConfig = ClientConfig
DataPA.Initialise()
```

Architect Syntax

```
DEFINE VARIABLE clientConfig AS DataPAClientConfig.ClientConfig.
DEFINE VARIABLE SessionData AS DataPA.SessionData.
DEFINE VARIABLE DataPAApplication AS DataPA.Application.

SessionData = NEW DataPA.SessionDataClass().
SessionData:SetComValue("DelayInitialise", "True").
clientConfig = NEW DataPAClientConfig.ClientConfig().

clientConfig:UserName = "Guest".
clientConfig:Password = "openanalytics".
clientConfig:UsePreviousUsernameAndPasswordSet = True.

DataPAApplication = NEW DataPA.ApplicationClass().
DataPAApplication:ClientConfig = clientConfig.
DataPAApplication:Initialise().
```

Setting Username and Password for individual systems

In certain circumstances, different systems configured within the analytics engine may require different authentication details. In these circumstances we can use the systems collection to configure the authentication details for each system independently.

The Systems Collection

DataPA represents AppServer connections with systems. Systems are accessible programmatically from the Systems property on the DataPA.Application object. The Systems property returns a collection of all the Systems available for this instance of DataPA.

AppBuilder Syntax

```
COM-hdl-var:Systems
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application object.

The Systems property returns a COM-Handle that represents the Systems collection.

Visual Studio Syntax

```
Application.Systems
```

Application
A variable that references a valid DataPA.Application object.

The Systems property returns a handle that represents the Systems collection.

Architect Syntax

```
Application:Systems().
```

Application
A variable that references a valid DataPA.Application object.

The Systems property returns a handle that represents the Systems collection.

The Systems collection has one property and one method that allow us to set and retrieve information about the systems. First is the Count property that retrieves the number of systems defined for this instance of DataPA.

AppBuilder Syntax

```
COM-hdl-var:Count
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Application.Systems object.

To retrieve individual systems, the systems object has an Item method. The Item method has the following syntax:

```
COM-hdl-var:item(expression, findstring)
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA.Query.RequiredFields object.

expression
An integer expression that resolves to the position of the system in the list of systems.

findstring
A string expression that resolves to a valid find string expression.

The Item method returns a COM-handle that represents a DataPA.Fields object.

For example, the following code displays the name of each system defined for the instance of DataPA:

```
DEFINE VARIABLE chSystem AS COM-HANDLE   NO-UNDO.
DEFINE VARIABLE ix AS INTEGER     NO-UNDO.

REPEAT WITH ix = 1 TO chDataPA:Systems:COUNT.
  chSystem = chDataPA:Systems:ITEM(ix, "").
  MESSAGE chSystem:NAME
    VIEW-AS ALERT-BOX INFO BUTTONS OK.
END.
```

The following code retrieves the com handle for the sports2000 system:

```
chSystem = chDataPA:Systems:ITEM( ,"cName='sports2000'").
```

Visual Studio Syntax

```
Systems.Count
```

Systems
A variable that references a valid DataPA.Application.Systems object.

To retrieve individual systems, the systems object has an Item method. The Item method has the following syntax:

```
Systems.item(expression, findstring)
```

Systems
A variable that references a valid DataPA.Systems object.

expression
An integer expression that resolves to the position of the system in the list of systems.

findstring
A string expression that resolves to a valid find string expression.

The Item method returns a handle that represents a DataPA.Fields object.

For example, the following code displays the name of each system defined for the instance of DataPA:

```
dim System As DataPA.System
dim ix     As Integer

for ix = 1 TO Systems.Count
  System = Application.Systems.ITEM(ix, "")
  msgbox System.NAME
next ix
```

The following code retrieves the handle for the sports2000 system:

```
System = Application.Systems.ITEM( ,"cName='sports2000'")
```

Architect Syntax

```
Systems:Count().
```

Systems
A variable that references a valid DataPA.Application.Systems object.

To retrieve individual systems, the systems object has an Item method. The Item method has the following syntax:

```
Systems:item(expression, findstring).
```

Systems
A variable that references a valid DataPA.Systems object.

expression
An integer expression that resolves to the position of the system in the list of systems. Pass “?” is not being used.

findstring
A string expression that resolves to a valid find string expression. Pass "?" is not being used.

The Item method returns a handle that represents a DataPA.Fields object.

For example, the following code displays the name of each system defined for the instance of DataPA:

```
DEFINE PRIVATE VARIABLE System AS DataPA.System NO-UNDO.
DEFINE PRIVATE VARIABLE ix     AS INTEGER       NO-UNDO.

REPEAT WITH ix = 1 TO Application:Systems:COUNT():
  System = Application:Systems:ITEM(ix,"?").
  MESSAGE System:NAME
    VIEW-AS ALERT-BOX INFO BUTTONS OK.
END.
```

The following code retrieves the handle for the sports2000 system:

```
System = Application:Systems:ITEM("?","cName='sports2000'").
```

The Username Property

The Username property on the system object allows the username to be set programmatically.

AppBuilder Syntax

```
COM-hdl-var:Username
```

COM-hdl-var
A COM-HANDLE variable that references a valid DataPA system object.

For example, the code to set the username for the sports2000 system to *Nick* could be:

```
chSystem = chDataPA:Systems:ITEM( ,"cName='sports2000'").
chSystem:Username = “Nick”.
```

Visual Studio Syntax

```
System.Username
```

System
A variable that references a valid DataPA system object.

For example, the code to set the username for the sports2000 system to *Nick* could be:

```
System = Application.Systems.ITEM( ,"cName='sports2000'")
System.Username = “Nick”
```

Architect Syntax

```
System:Username().
```

System
A variable that references a valid DataPA system object.

For example, the code to set the username for the sports2000 system to *Nick* could be:

```
System = Application:Systems:ITEM(“?”,"cName='sports2000'").
System:Username = “Nick”.
```

The Password Property

The Password property on the system object allows the password be to set programmatically.

AppBuilder Syntax

```
COM-hdl-var:Password
```

COM-hdl-var
A COM-HANDLE variable that references a valid system object.

For example, the code to set the username for the sports2000 system to *Nick* and the password to "1234" could be:

```
chSystem = chDataPA:Systems:ITEM( ,"cName='sports2000'").
chSystem:Username = "Nick".
chSystem:Password = "1234".
```

Visual Studio Syntax

```
System.Password
```

System
A variable that references a valid system object.

For example, the code to set the username for the sports2000 system to *Nick* and the password to "1234" could be:

```
System = Application.Systems.ITEM( ,"cName='sports2000'")
System.Username = "Nick"
System.Password = "1234"
```

Architect Syntax

```
System:Password.
```

System
A variable that references a valid system object.

For example, the code to set the username for the sports2000 system to *Nick* and the password to "1234" could be:

```
System = Application:Systems:ITEM("?","cName='sports2000'").
System:Username = "Nick".
System:Password = "1234".
```

The PromptUser property

The PromptUser property on the system object determines whether or not the user is prompted for a username and password when DataPA connects to the AppServer.

AppBuilder Syntax

```
COM-hdl-var:PromptUser
```

COM-hdl-var
A COM-HANDLE variable that references a valid system object.

For example, the code to set the username for the sports2000 system to *Nick* and the password to "1234" and to set the PromptUser property to false so the user is not prompted could be:

```
chSystem = chDataPA:Systems:ITEM( ,"cName='sports2000'").
chSystem:Username = "Nick".
chSystem:Password = "1234".
chSystem:PromptUser = FALSE.
```

Visual Studio Syntax

```
System.PromptUser
```

System
A variable that references a valid system object.

For example, the code to set the username for the sports2000 system to *Nick* and the password to "1234" and to set the PromptUser property to false so the user is not prompted could be:

```
System = Application.Systems.ITEM( ,"cName='sports2000'")
System.Username = "Nick"
System.Password = "1234"
System.PromptUser = FALSE
```

Architect Syntax

```
System:PromptUser.
```

System
A variable that references a valid system object.

For example, the code to set the username for the sports2000 system to *Nick* and the password to "1234" and to set the PromptUser property to false so the user is not prompted could be:

```
System = Application:Systems:ITEM("?","cName='sports2000'").
System:Username = "Nick".
System:Password = "1234".
System:PromptUser = FALSE.
```

Using DataPA Application Events

This lesson will teach the how to use DataPA Application events from your third party application.

Learning Objectives

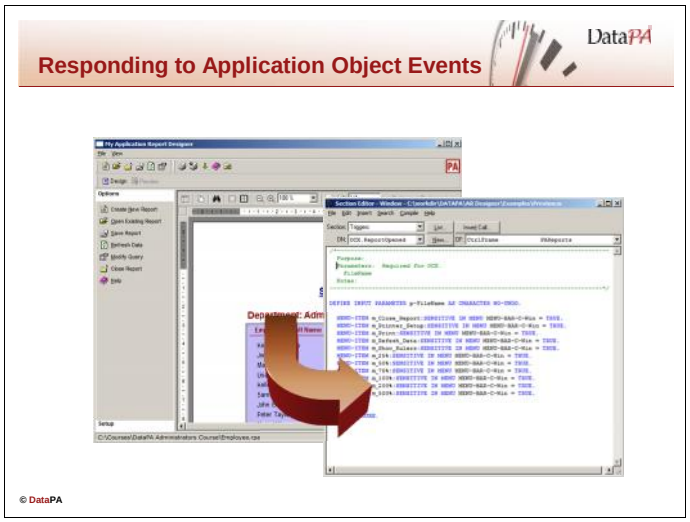
When you complete this lesson you should be able to:

- Add event handlers to you application to respond to the events raised by the DataPA Application objects.

Prerequisites

Before you begin this lesson you should be able to:

- Create an application in the development tool you have chosen
- Use the DataPA Reports control and DataPA Application objects in the application you have created.



Responding to Application Object Events in the AppBuilder

Progress handles Application Object events using event procedures. An *event procedure* is a standard Progress internal procedure that serves as an event handler for ActiveX objects. The Application objects will often pass one or more parameters that must be handled by the event procedure. These parameters are defined as parameters in the event procedure. Progress identifies an event procedure from the construction of its name. This is the only syntactic feature that distinguishes Progress internal procedures as an event procedure.

Creating Event Procedures in the AppBuilder

By default the events are turned off for COM objects in the AppBuilder. They must be enabled for each COM object individually using a special command in the Progress ABL.

So consider the case where a COM handle to a DataPA Application object is stored in the variable chDataPA. To enable events for this Application object you would use the method below and pass the method an identifying name which for the case below we have used “DataPA”:

```
chDataPA:ENABLE-EVENTS("DataPA").
```

Having enabled the events follow these steps to create an event procedure in the AppBuilder:

- First you need to identify if the events procedure will require any parameters. The best way to do this is to use a tool that is included in the Progress installation. This is called the COM Object Viewer and will probably not be on the Start menu under Progress. If can be found in the bin folder of your Progress installation and is called proobjvw.exe. It’s icon is  . This tool will allow you to browse to the COM object (for example DataPA.dll) and see the properties, methods and events. The good thing about it is that it will provide the correct Progress syntax for you. For example for the DataAvailable event of the DataPA Application object:

```
PROCEDURE <event-proc-prefix>.DataAvailable :

END PROCEDURE.
```

An example with some parameters passed in the event would be the RunQuery event of the DataPA Application object:

```
PROCEDURE <event-proc-prefix>.RunQuery :
       DEFINE INPUT-OUTPUT PARAMETER Query AS COM-HANDLE.
       DEFINE INPUT-OUTPUT PARAMETER Cancel AS LOGICAL.

END PROCEDURE.
```

NB: The <event-proc-prefix> will be the name you gave when calling the ENABLE-EVENTS method to enable events on the COM object.

- Insert the code from the COM Object Viewer into your AppBuilder program or window and program the logic that you want to before on execution of the event.

Responding to Events in Visual Studio

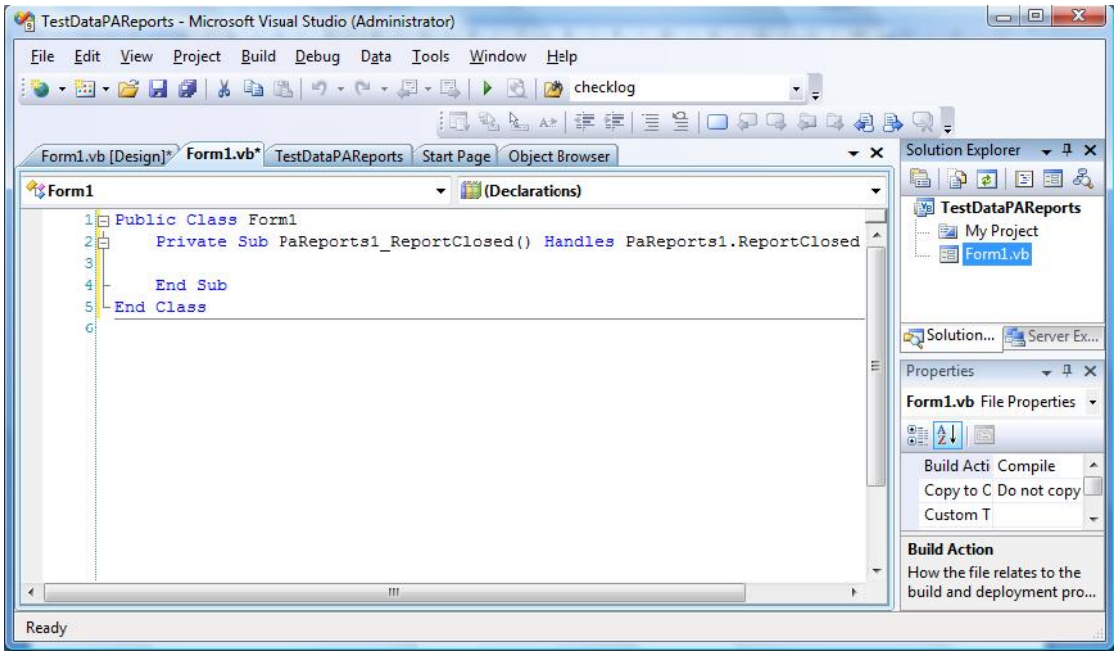
Visual Studio handles Application object events using standard .NET event procedures. The application objects will often pass one or more parameters that must be handled by the event procedure. These parameters are defined as parameters in the event procedure. Visual Studio identifies an event procedure from handler that is assigned to it for example (`Handles PAApp.DataAvailable`).

Creating Event Procedures in Visual Studio

Follow these steps to create an event procedure in Visual Studio:

- Ensure that the variable defining the Application object is defined with the WithEvents option
- Select the View Code option for the form
- Select the Application instance from the object drop down
- Select the event from the drop down list of events beside it

Visual Studio will create an appropriate event procedure with the correct handler assigned.



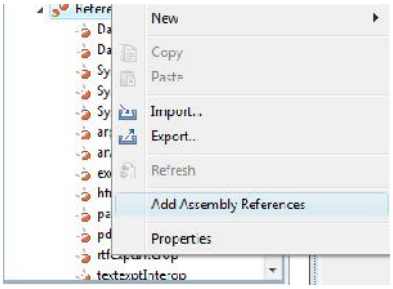
Responding to Events in Architect

Unlike Visual Studio, OpenEdge Architect does not support events for objects which do not have a fixed standard interface of Sender and EventArgs. This means that the events generated by the DataPA Application objects cannot be picked up directly by Architect.

Creating Event Procedures in Architect

In order to work round this limitation in OpenEdge Architect a supplementary assembly has been created by DataPA called DataPAEventHandlerLib.dll. This assembly is installed in the GAC and picks up events from the DataPA Application objects and generates them in an OpenEdge Architect friendly form.

To reference the DataPAEventHandlerLib assembly, expand your project node in the treeview on the right hand side of Architect, select the Referenced Assemblies node and press the right mouse button. Select the Add Assembly Reference and you will be presented with a list of assemblies available on your machine.



Finally, select the DataPAEventHandlerLib from the list and click OK. The DataPAEventHandlerLib assembly to has now been added to your project.

Once the DataPAEventHandlerLib assembly has been added to your project, you need to create one or more instances of the Event handler. There are a number of Event handler types which allow you access to the events of their respective DataPA Application objects. These are listed below.

DataPA Object	DataPAEventHandlerLib class
Application	DataPAEventHandlerLib.Application
PAConnection	DataPAEventHandlerLib.PAConnection
Query	DataPAEventHandlerLib.Query
Security	DataPAEventHandlerLib.Security

Once created, you need to assign the Event Handler object to the object it is going to expose the events of. The following table gives examples on how to do this with each of the event handler objects.

DataPA Object	DataPAEventHandlerLib Property
`Application`	`PAAppEvent:Application = PAApp.` Where: ***PAApp*** is the DataPA Application object accessed from the Reports control. ***PAAppEvent*** is the instance of the DataPAEventHandlerLib.Application class created
`PAConnection`	`PAConEvent:PAConnection = PAConnection.` Where: ***PAConnection*** is the DataPA Connection object accessed from the DataPA Application object. ***PAConEvent*** is the instance of the DataPAEventHandlerLib.PAConnection class created
`Query`	`PAQueryEvent:Query = PAQuery.` Where: ***PAQuery*** is the DataPA Query object accessed from the DataPA Application object. ***PAQueryEvent*** is the instance of the DataPAEventHandlerLib.Query class created
`Security`	`PASecEvent:Security = PASecurity.` Where: ***PASecurity*** is the DataPA Security object accessed from the DataPA Security object. ***PASecEvent*** is the instance of the DataPAEventHandlerLib.Security class created

We now need to define and reference a procedure to handle each required event of the DataPAEventHandlerLib assembly. Architect enables an event handler procedure using the subscribe statement and this must be explicitly called to route the event to the event handler procedure. For example, to assign an event handler procedure for the DataAvailable event we would use the command below:

```
PAAppEvent:DataAvailable:Subscribe(DataAvailableHandler).
```

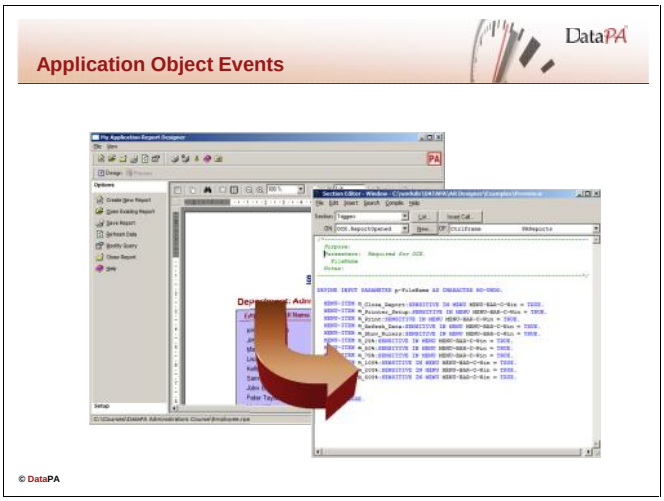
Where:
DataAvailableHandler is the event handler procedure in your Architect class.
PAAppEvent is the instance of the DataPAEventHandlerLib.Application class created.

Finally we need to create the event handler procedure. The following example shows an event handler procedure that responds to the DataAvailable event.

```
METHOD PRIVATE VOID dataavailablehandler
( INPUT sender AS System.Object, INPUT e AS System.EventArgs ):
  /* Your event logic */
  RETURN.
END METHOD.
```

The following example shows how to create a DataPAEventHandlerLib instance to hook into the DataAvailable event of a DataPA Application object (PAApp) and set a handler for that DataAvailable event:

```
DEFINE PRIVATE VARIABLE PAAppEvent AS DataPAEventHandler.Application.
PAAppEvent = NEW DataPAEventHandler.Application().
PAAppEvent:Application = PAApp.
PAAppEvent:DataAvailable:Subscribe(DataAvailableHandler).

METHOD PRIVATE VOID dataavailablehandler
( INPUT sender AS System.Object, INPUT e AS System.EventArgs ):
  /* Your event logic */
  RETURN.
END METHOD.
```



Introduction

The DataPA Application objects provide events to allow the developer to respond to changes in the state of each of these objects.

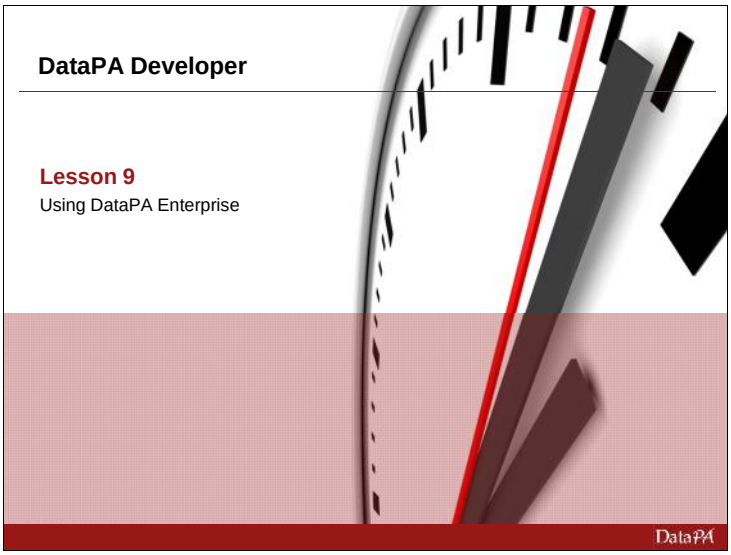
DataPA Application Object Events

These events are listed in the table below:

DataPA Object	Event Name	Parameters	Definition & Usage
Application	DataAvailable	none	This event is fired when data is available after the running of a query. This can be used to tell when a query has completed.
Application	QueryOpened	Query (Query)	This event is fired when a query is opened and the Query object is passed in the Query parameter of the event.
Application	QueryClosed	None	This event is fired when a query is closed.
Application	CancelQueryWizard	None	This event fires when the Query Wizard is cancelled.
Application	CancelLinkWizard	None	This event fires when the Link Wizard is cancelled.
Application	RunQuery	Query (Query) Cancel (Boolean)	This event fires when a Query is about to be run. The parameters are the Query object of the Query that is about to be run and a Cancel flag that allows the run of the Query to be cancelled by setting the Cancel flag to TRUE.

DataPA Object	Event Name	Parameters	Definition & Usage
Application	SetupRefreshed	None	This event fires when the DataPA Setup is refreshed. This means that the configuration information has been read and may have changed.
Application	CloseQueryRequest	paResponse	This event fires when a query is about to close. The parameter passed back is paResponse and has the following values: paNoResponse = 0 paUserRespondedNoClose = 1 paUserRespondedOKClosed = 2
Application	LicenseChanged	none	This event fires when the license that is used within DataPA changes.
Application	SlientSet	Silent (Boolean)	This event fires when the Silent property of the Application object has been changed. The changed value is passed in the bSilent parameter.
Application	QueryAlive	Cancel (Boolean)	This event fires every second when a query is being run. This allows the programmer to perform so action such as updating a progress bar or changing an animation. The Cancel parameter allows the programmer to cancel a running query if required.
PAConnection	BeforeDisconnect	bCancel (Boolean)	This event fires before the AppServer is disconnected. The bCancel parameter allows the programmer to cancel this disconnect action if required.
PAConnection	AfterDisconnect	None	This event fires after the AppServer is disconnected.
PAConnection	BeforeConnect	bCancel (Boolean)	This event fires before the AppServer is connected. The bCancel parameter allows the programmer to cancel this connect action if required.
PAConnection	AfterConnect	None	This event fires after the AppServer is connected.

DataPA Object	Event Name	Parameters	Definition & Usage
PAConnection	LoginInformationRequest	LoginInfo (LoginInfoEventArgs)	This event fires when the user is going to be presented with the DataPA Login screen and be asked to provide credentials to connect to the AppServer with. The parameter LoginInfo contains an instance of the LoginInfoEventArgs object which provides details of the user login information. This allows the programmer to override the login screen and provide the login details programmatically.
Query	CancelRunQuery	None	This event fires when the query that is being run is cancelled.
Security	RequestServerSecData	None	This event fires when DataPA makes a request for Security information from the AppServer. This can only occur when the data location is setup to be an AppServer.
Security	LicenseChanged	None	This event fires when the license that is used with DataPA changes.



Using DataPA Enterprise

This lesson will teach the how to use DataPA Enterprise from your third party application.

Learning Objectives

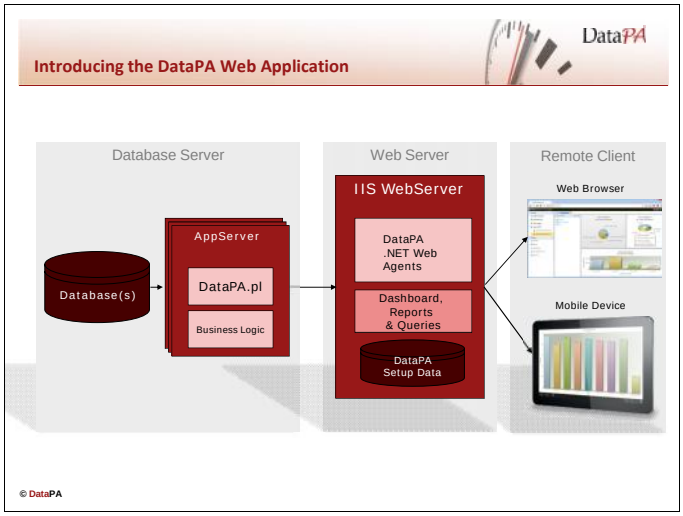
When you complete this lesson you should be able to:

- Embed dashboards, reports and queries in your web application
- Pass authentication details to DataPA Enterprise from your web application.
- Pass parameters to dashboards, reports and queries from your web application.

Prerequisites

Before you begin this lesson you should be able to:

- Create an application in the development tool you have chosen
- Execute a URL string from your application
- Use iframes to embed 3rd party content in your chosen application.



Introducing the DataPA Web Application

The DataPA Web Application, a component of DataPA Enterprise, is a Web Application that allows business users to run dashboard, reports and queries across a wide area network or the Internet, using any standard web browser. Based on Microsoft ASP.NET technology, the DataPA Web Application can be hosted on any compatible IIS Web Server, and connects to your database(s) and runs queries via the AppServer. The Web Application is designed to allow content to be embedded into 3rd party applications by rendering it independently in a browser, or as part of another we page in an iFrame.

DataPA Web Application Architecture

To refresh a dashboard, report or query using the DataPA Web Application three key components are required. These components are usually distinguished by the physical machine on which they reside (although all three could reside on a single machine, or any combination thereof, and the database server could reside across many machines). The three components are the Database server, the Web Server and the Remote Client, and will be described in detail in the following sections.

The Database Server

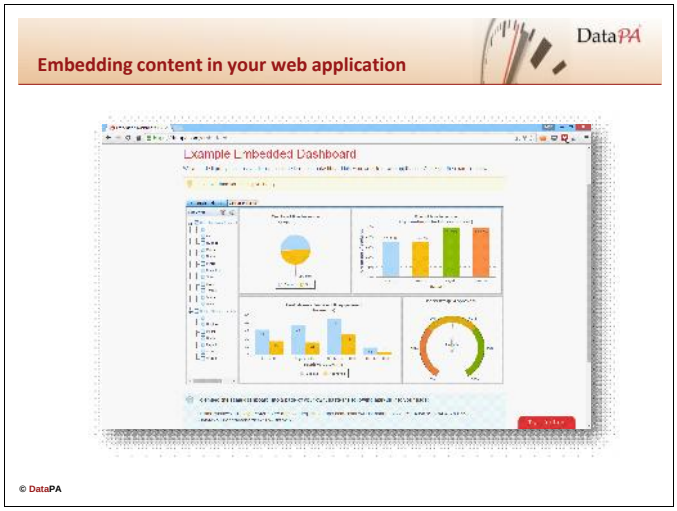
The database server comprises of the database (or databases) and a Progress AppServer configured for DataPA. With the exception of an optional business logic procedure to filter reports for the web front end, described in the DataPA Administration course, the AppServer required for the DataPA Web Application is configured in exactly the same way as the AppServer for the standard DataPA applications. Indeed, it is often the case that a single AppServer will service both standard and web DataPA users.

The Web Server

The web server comprises of a Microsoft Internet Information Server, with a configured web server, running the DataPA ASP.NET web application. The dashboard, reports, queries and setup data that is required to run these reports should reside on this server. Each Agent in the DataPA ASP.NET Web Application is an instance of DataPA that services client requests, has its own connection to the AppServer and is capable of rendering dashboards, reports and queries for web and mobile devices

The Remote Client

The Remote Client consists of any machine with a browser that supports JavaScript and HTML5 or a mobile device with the relevant DataPA Mobile app installed.



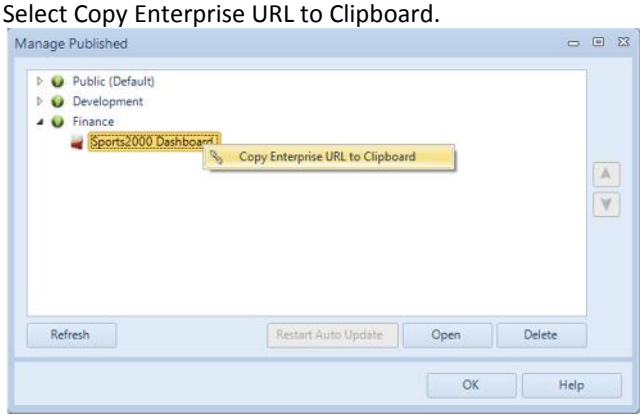
Embedding content in your web application

The web application component of DataPA Enterprise is designed to allow content to be embedded into 3rd party applications. As such, a simple mechanism is provided to allow developers to obtain the required URL to render content independently in a browser, or as part of another we page in an iFrame.

Obtaining the URL to embed content

Follow these steps to obtain the URL for a dashboard;

1. Open the DataPA Enterprise Dashboard client application.
2. From the Application Settings ribbon tab, select Manage Published.
3. Select to the dashboard, report or query you wish to embed in the treeview and press the right mouse button.
4. Select Copy Enterprise URL to Clipboard.



If your implementation of DataPA Enterprise requires authentication, the URL provided will be suffixed with the following;

&autoLogin=True&username=[Username]&password=[Password]

The username and password will need to be passed in the request, otherwise the application will redirect the user to the login screen. However, these value pairs are included for information only. In reality, **the application will not accept the username and password** in the URL (GET), they must be passed in the message body (POST). This ensures user login details are encrypted over a secure network and never passed as clear text.

Passing Parameters to DataPA Enterprise

You may wish to pass a value from your web application to DataPA Enterprise to filter the data shown in a dashboard before it is rendered. The steps to achieve this can be broken down into the following tasks, which will be described in detail in later in the lesson;

1. Create a dashboard with at least one required field query parameter.
2. Remove the control panel objects that provide the values for the required fields.
3. Set the query to refresh on open.
4. Publish the dashboard.
5. From the Manage Published screen, copy the URL and substitute the parameter and security parameters.
6. Add a hyperlink or iFrame definition in your web application to open the URL.

Managing Query Parameters

By default query parameters are added to a dashboard as a control panel object. The parameter values used to filter the query when it is refreshed are taken from these control panel objects, and the queries are refreshed when the user enters or selects a different value in this control.
However, in this case we want to provide the query parameter values from the third party web application, rather than the user controlling these values from an object on the dashboard.

Providing Query Parameters Values from an Application

When a query in a dashboard is refreshed by the web application it checks if there is a valid control panel object for each required field in the query. For each required field that does not have a valid control panel object, the web application attempts to retrieve the query parameter value from the URL.

Deleting Control Panel Objects

Follow these steps to remove control panel objects that provide query parameter values from the dashboard.

1. Open the dashboard in the DataPA Enterprise Dashboard application.
2. Double click on the control panel in the dashboard to open the Control Panel dialog box.
3. Select the control panel object in the control panel.
4. Select the Delete button in the ribbon.
5. Press OK

6. Save the dashboard.

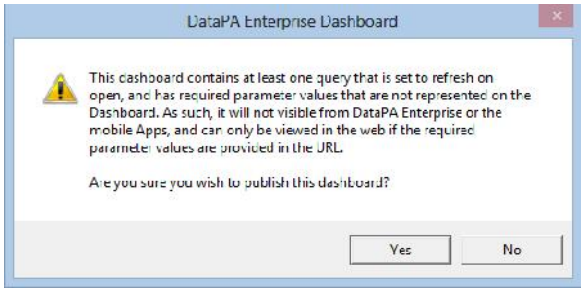
Set the query to refresh on open

To ensure the data in the dashboard is refreshed before it is opened, we must specify that the dashboard should be refreshed on open. Follow these steps to achieve this;

1. With the dashboard open in the DataPA Enterprise Dashboard application select *Manage Schedule* from the *Queries* group on the *Edit Dashboard* tab of the ribbon.
2. Ensure the *Refresh query when dashboard is opened* option is checked for the query with the required field.

Publish the dashboard

With the dashboard open in the DataPA Enterprise Dashboard application, select *File →Publish*. You will receive a message warning that the dashboard will not be available to the mobile Apps, and can only be viewed in the web if the required value is passed in the URL;



Obtaining the URL to embed content

Follow these steps to obtain the URL for a dashboard;

1. Open the DataPA Enterprise Dashboard client application.
2. From the Application Settings ribbon tab, select Manage Published.
3. Select to the dashboard, report or query you wish to embed in the treeview and press the right mouse button.
4. Select Copy Enterprise URL to Clipboard.

Along with the authentication details, the URL will also contain URL value pairs for any required field parameters. For instance (http://jsfiddle.net/pg7Lap3h/);

http://enterprise.datapa.com/DataPA/Dashboard.aspx?ID=5163daee-0222-4529-8074-0734fe3505d2&CustomerNumber=[VALUE]

In this example, we would replace the trailing [VALUE] with a value. For instance;

http://enterprise.datapa.com/DataPA/Dashboard.aspx?ID=5163daee-0222-4529-8074-0734fe3505d2&CustomerNumber=1