iFIX 6.1

Using the .NET Component

We want to hear from you. If you have any comments, questions, or suggestions about our documentation, send them to the following email address:

doc@ge.com

Table of Contents

Using the .NET Component

The .NET Component feature for iFIX allows you to host .NET components within your iFIX pictures. These .NET Components include pre-selected ones from the .NET Framework, samples from GE, or custom ones that you develop or buy.

Creating custom components requires proficiency in Microsoft® Visual Studio® 2010 and the .NET Framework. In addition, to develop your own .NET components, you must be proficient in programming with the .NET Framework. For iFIX scripting, you must have a working knowledge of the Visual Basic programming language.

To help you use the .NET Component feature, this Help file includes the following sections:

- Introduction
- General Overview of Component Hosting
- Advanced Features

Introduction

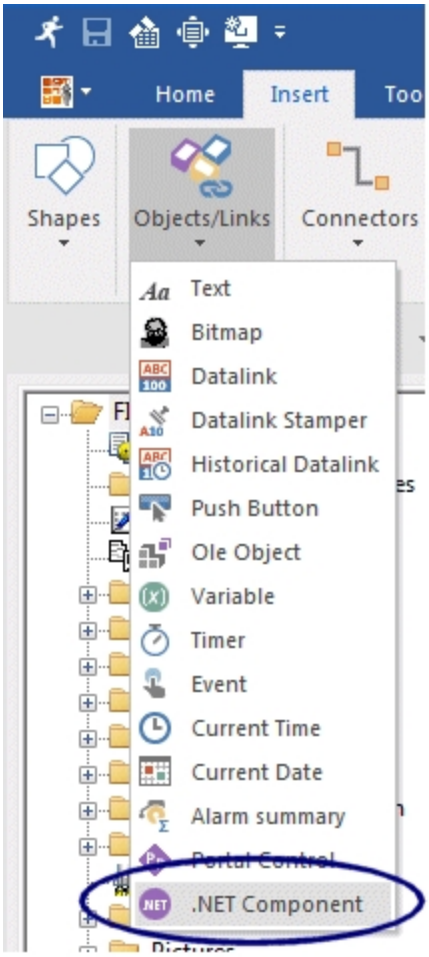
The .NET Component feature allows you to host .NET components within your iFIX pictures. All hosted .NET components will look and function as OLE objects (ActiveX controls) within your iFIX pictures. Just as other iFIX controls allow you to access the Property Window and Basic Animations dialog box, so do hosted .NET components. Screen editing actions, such as Select, Move, Resize, Delete, Undo, Copy, and Paste are all supported for .NET components as well.

The Component Hosting feature is installed by default when you install iFIX.

NOTES:

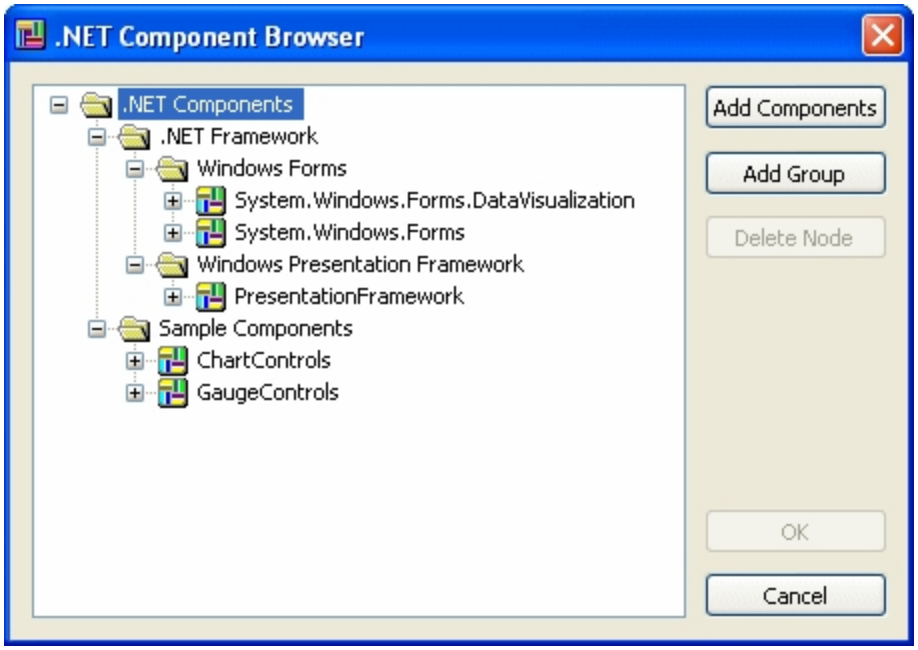
- The Backup and Restore Utility does not support backing up .NET Component files.
- .NET Component is not supported on the XP Embedded OS.
- Upgrading or in any way re-installing iFIX will result in replacing any .NET Component files that are supplied with the product install. This includes sample files such as Visual Studio solutions and projects in the Sample Components folder which you may have modified. If you have modified these files, they will lose their changes on an upgrade or re-install of iFIX. If you make any changes to these Visual Studio files, they should be backed up prior to upgrade/re-install.
- If a user is a member of the Power Users group in Windows security, that user will be unable to open pictures with .NET components in them unless an Administrator has previously opened that picture on that PC. When an Administrator opens the picture they register the .NET assemblies for the contained controls on that machine, which allows Power Users to later open that same picture. Standard Windows Users do not encounter this same problem. This note applies to Power Users in Windows Vista and later.

You can access the .NET Component on the Insert menu in the iFIX WorkSpace, as shown in the following figure.



General Overview of Component Hosting

You can access the .NET Component on the Insert tab in the iFIX WorkSpace by selecting Objects/Links, and then clicking .NET Component. The .NET Component Browser dialog box appears, as shown in the following figure.

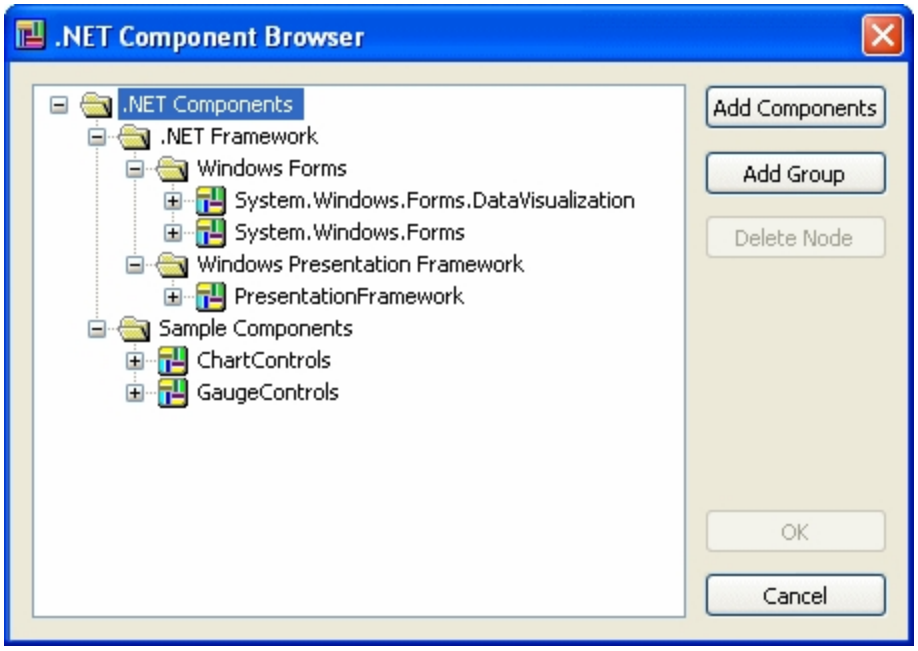


The hosting environment:

- Exposes the public properties, methods, and events of the underlying .NET control.
- Provides type descriptions to support property editing and binding, method invocation, and event handling with the iFIX user interface and scripting.

.NET Component Browser Dialog Box

The .NET Component Browser dialog box allows you to manage .NET components for use within your iFIX pictures. After selecting it from Objects/Links on the Insert tab in the iFIX WorkSpace, the .NET Component Browser dialog box appears as shown in the following figure.



The .NET Component Browser dialog box provides the following capabilities:

Browse Tree

The tree allows you to browse the .NET components installed on your computer and in the sample projects folder.

Add Components

Click to add a new set of components from a .NET assembly to the list of components in the browser.

Add Group

Click to add a new group in the browser.

Delete Node

Select an Assembly node or an empty Group node and click Delete to remove it from the browse tree.

Help

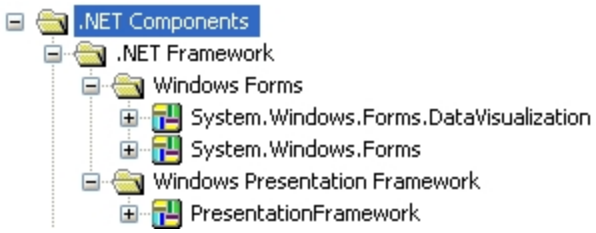
Click to display this Help file.

Supported .NET Control Types

The components initially listed in the .NET Component Browser dialog box are built-in components that can be used immediately. These supported .NET control types describe the kinds of .NET classes

(types) that are able to be hosted; this is intended to guide advanced users by indicating the kinds of .NET controls they can develop or buy from a third party.

Component hosting in iFIX supports both Windows Forms (WinForms) and Windows Presentation Framework control types in the Microsoft .NET Framework. You can find these built-in controls in the .NET Component Browser dialog box. One group is labeled "Windows Forms," and the other group is named "Windows Presentation Framework." The Windows Forms group contains the System.Windows.Forms.DataVisualization and System.Windows.Forms assemblies. The Windows Presentation Framework group contains the PresentationFramework assemblies. The following figure illustrates these groups and assemblies.



The individual controls contained in each of these subfolders cover the majority of .NET controls used for non-web-based applications.

Inserting a .NET Component into a Picture

You can access the .NET Component feature on the Insert menu in the iFIX WorkSpace. This menu option opens the .NET Component Browser dialog box, allowing you to select a .NET component to insert into your picture. After you insert the component into your picture, you can modify the properties and then bind the component to a data source to any of these properties.

To insert a .NET component into a picture:

1. In the iFIX WorkSpace, in Ribbon view, click the Insert tab.
2. On the Insert tab, select Objects/Links, and then click .NET Component. The .NET Component Browser dialog box appears.
3. In the tree browser, select the .NET component that you want to add and then click OK. (A component must be selected for the OK button to be enabled. For example, you could add one of the two sample components: TrendChart or LinearGauge.)

To access the properties for the .NET component:

1. In the iFIX picture, select the .NET Component for which you want to view the properties.
2. Right-click the component and select Property Window. The Property Window appears, if it is not already displaying.
3. Scroll through the Alphabetic list to locate the property that you want to view or modify.

Binding .NET Component Properties to an iFIX Data Source

The properties exposed by a .NET component, as listed on the Property Window, can be bound to iFIX data sources. As a result of a property binding, the value of the property will change once the value of the data source changes. This makes the component a monitor of the property value.

To bind a .NET component to an iFIX data source:

1. In the iFIX WorkSpace, right-click the .NET component and select Animations. The Basic Animations Dialog appears.
2. Under Advanced Animations, click the Configure button. The .NET Component Animations dialog box appears.
3. On any of the tabs, locate the property that you want to bind to a data source, and then click the Animate check box next to the property. The Data Source area appears below the list of properties (if it is not already displaying).
4. Next to the Data Source field, click the Browse (...) button. The Expression Builder dialog box appears.
5. Select a data source and click OK.
6. Enter the Animation or Historical Properties and click OK. For more information on creating data sources, refer to the *Creating Pictures* e-book.

Example

The following example shows how to set the Reading property of the LinearGauge sample object to a Float data type that has a value range from 0 to 100, and a data source of the current value. This example assumes that you already have a RAMP block created within the iFIX database.

1. In your iFIX picture, insert the sample LinearGauge .NET component.
2. Right-click the component and select Animations. The Basic Animations Dialog appears.
3. Under Advanced Animations, click the Configure button. The .NET Component Animations dialog box appears.
4. On the Component tab, locate the Reading property in the property list.
5. Click the Animate check box next to the Reading property.
6. Next to the Data Source field, click the Browse (...) button. The Expression Builder dialog box appears.
7. Select a configured data source of the Float type that has a value range from 0 to 100. For example, if you have a RAMP block already created in your database, select the RAMP.F_CV data source.
8. On the Animation Properties tab, in the Data Conversion drop-down list box, select Object.
9. Click the OK button. After the dialog boxes close, the Save button gets enabled because a data binding has been created and added to the current picture.
10. Click the Save button to save the current page.
11. Click the Run button on the top toolbox of the iFIX WorkSpace to switch the picture into the run mode and view the Linear Gauge object with its pin handle moving as the data source changes.

Using Font and Enumeration Properties for .NET Components

The Font and Enumeration properties for .NET components can be accessed from the iFIX Property Window and through VBA scripting. For example, the Brush property in the Windows Presentation Framework is split into individual fields (colors and positions), so that it may be easily configured within the iFIX Property Window editor. The Font and Enumeration properties are also converted for the same convenience.

For information on working with font and enumeration properties of .NET Components, refer to the following sections:

- Font Properties for Windows Forms Components
- Font Properties for WPF Components
- Enumeration Properties for .NET Components

Font Properties for Windows Forms Components

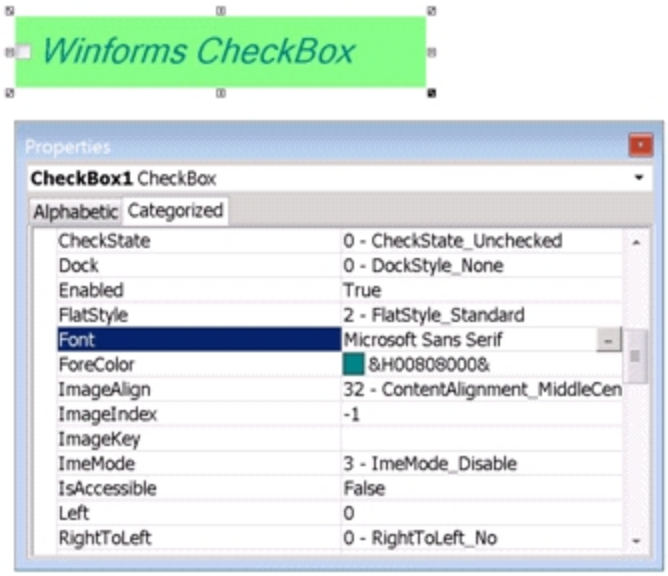
Windows forms-based controls use the Font class (System.Drawing.Font) to access the font element in painting their graphics. The Font class is converted to an interface that is compatible to the VBA IFontDisp interface. As a result, any font properties of Windows forms-based controls may be configured the same way ActiveX controls are configured in the iFIX WorkSpace.

To access font properties for Windows Forms components from the iFIX WorkSpace:

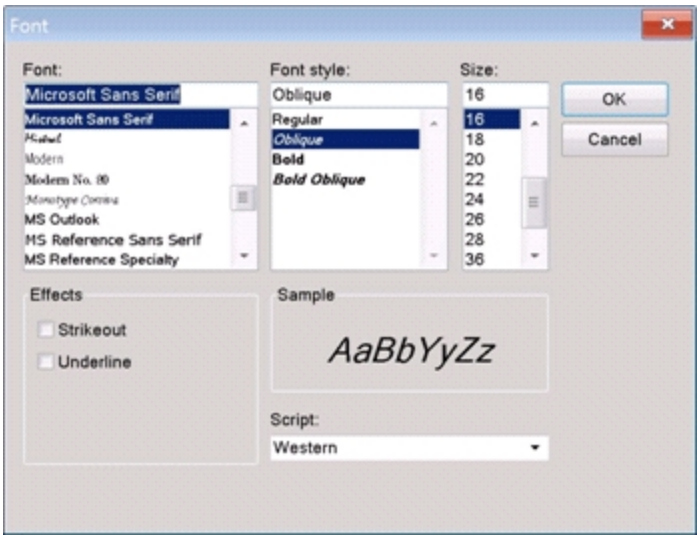
1. In the iFIX picture, select the .NET Component for which you want to access the font properties.
2. Right-click the component and select Property Window. The Properties window appears, if it is not already displaying.
3. Scroll through the Alphabetic list to locate the font property that you want to view or modify.
4. Click the ellipsis (...) button next to the font property. The Font dialog box appears.
5. Enter your changes and click OK.

Example

The following figure shows a Windows Forms CheckBox control (in the .NET Component Browser dialog box: .NET Components > .NET Framework > Windows Forms > System.Windows.Forms > CheckBox) and its Font property highlighted on the Property Window dialog box of the iFIX WorkSpace.



Click the ellipsis (...) button, and the Font dialog box appears, as shown in the following figure. This dialog box allows for setting a font by selecting its type, style, size, and effects.



Font Properties for Windows Presentation Framework Components

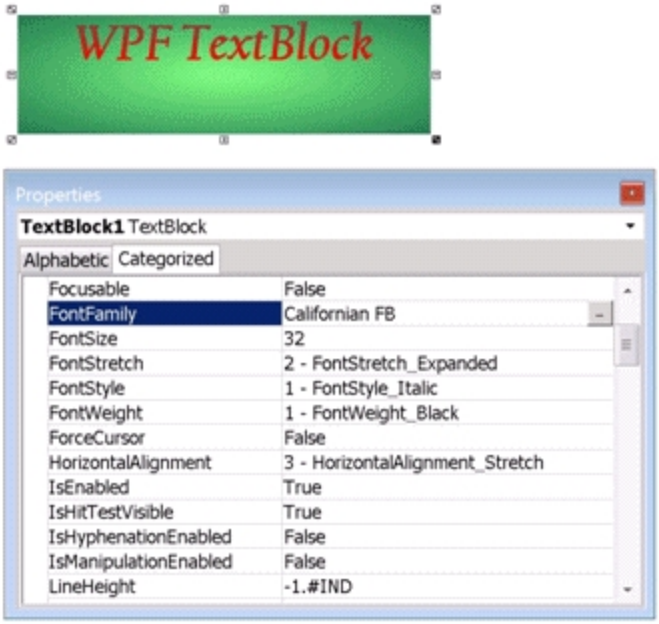
Windows Presentation Framework controls follow a different convention for font properties from Windows Forms (WinForms) controls. For Windows Presentation Framework controls, the FontFamily class only includes information for a font type. Other font structures, such as FontStretch, FontStyle, and FontWeight, handle the various font styles and effects. For these controls, the size of a font is accessed through a separate property usually called FontSize. As the Font dialog box is still the best way to select from all available fonts, the FontFamily data type is converted to the IFontDisp compatible interface.

To access font properties for Windows Presentation Framework components from the iFIX WorkSpace:

1. In the iFIX picture, select the .NET Component for which you want to access the font properties.
2. Right-click the component and select Property Window. The Property Window appears, if it is not already displaying.
3. Scroll through the Alphabetic list to locate the font property that you want to view or modify.
4. Click the ellipsis (...) button next to the font property. The Font dialog box appears.
5. Enter your changes and click OK.

Example

The following figure shows a Windows Presentation Framework TextBlock control (in the .NET Component Browser dialog box: .NET Components > .NET Framework > Windows Presentation Framework > PresentationFramework > TextBlock) and its Font property highlighted on the Property Window dialog box.



Click the ellipsis (...) button, and the Font dialog box appears. Be aware that unlike the Font dialog box for the Windows Forms components, only a font type can be set within the Font dialog box for Windows Presentation Framework controls. Other font properties can be set directly through separate font properties. For a TextBlock component, as shown in the previous figure, the properties are named: FontSize, FontStretch, FontStyle, and FontWeight.

NOTE: Not all fonts support all FontStretch, FontStyle, and FontWeight settings.

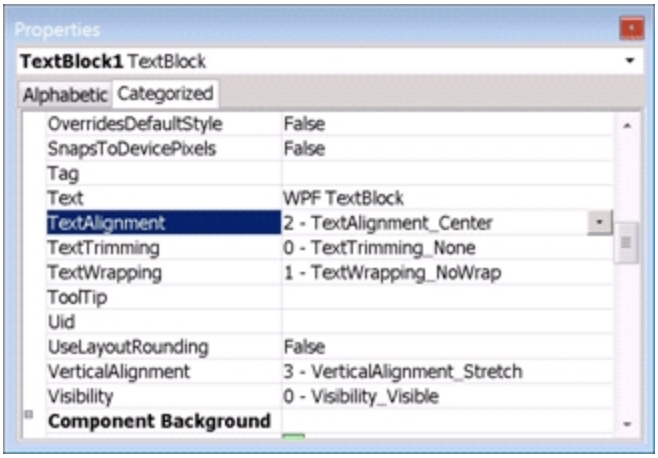
Enumeration Properties for .NET Components

All enumerations are internally numeric types, but enumeration types make property access easier as they provide meaningful names for the underlying numeric values. For example, for a horizontal alignment property, instead of setting somewhat meaningless and error-prone numbers such as 0, 1, 2, and so on, you can select a setting name such as Left, Center, and Right from the available setting list.

All the .NET Framework enumeration types used in .NET Components are converted to VBA-recognizable enumerable types. Enumeration values are shown on the Property Window editor, following the VBA convention, as:

<numeric value> - <enumeration type name>_<enumeration name>

The following figure shows some enumeration properties of the Windows Presentation Framework TextBlock control in the iFIX Property Window editor.



Scripting in VBA

The Property Window editor and the Basic Animation dialog box in the iFIX WorkSpace allow for configuration and binding of properties of primitive data types. Beyond these, the integrated VBA scripting supported within iFIX allows for:

- Getting or setting properties of the primitive and complex data types.
- Calling methods.
- Handling events.

Because the hosting environment for the .NET Component exposes all the properties, methods, and events of the underlying .NET component with appropriate data conversions, the underlying component can be fully manipulated with VBA scripting.

The following sections describe how to access these features in VBA:

- Adding References in VBA
- Using Intellisense
- Accessing Component Properties and Methods Through Scripting
- Using Event Handlers
- Handling Events with Non-Converted Parameters
- Using Properties and Methods of the iFIX Container

Adding References in VBA

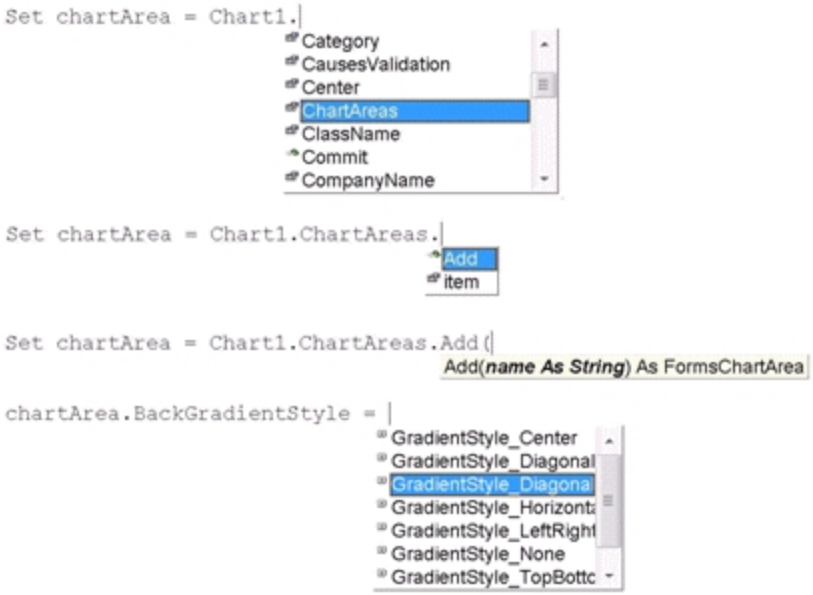
The GEIP_Orion_DataConversion reference must be added to the project in the Microsoft Visual Basic project in order to access .NET Component properties for non-primitive data types through scripting. Optionally, you may also want to include the Microsoft ActiveX Data Objects Library (version 2.7 or higher) or other references.

To add references in Microsoft Visual Basic:

1. In the iFIX WorkSpace, right-click the component and select Edit Script. The Microsoft Visual Basic Editor appears.
2. On the Tools menu, click References. The References dialog box appears.
3. Scroll through the list of components and make sure check marks appear next to the references you want to add. For instance:
    - GEIP_Orion_DataConversion (for the .NET Component)
    - Microsoft ActiveX Data Objects Library (version 2.7 or higher for other references)
4. Click OK.

Using Intellisense®

VBA scripting is assisted with IntelliSense because the hosting environment exposes the properties and methods of the underlying .NET component with data types that VBA scripting supports. The following figure shows IntelliSense examples at work for the Chart object included in the Windows Forms folder.



Accessing Component Properties and Methods Through Scripting

Some properties and methods unavailable through the Property Window in the iFIX WorkSpace (because they involve complex data types) can be accessed through scripting. To access scripting in iFIX, select the component, and on the right-click menu, select Edit Script to open the Visual Basic Editor. Before you begin editing your scripts, you must add the proper references (for GEIP_Orion_ DataConversion and Microsoft ActiveX Data Objects Library, version 2.7 or higher, if required). For steps, see the Adding References in VBA section.

Example

This example takes the Chart component from the Windows Forms (in the .NET Component Browser dialog box: .NET Components > .NET Framework > Windows Forms > System.Windows.Forms.DataVisualization > Chart), and adds a ChartArea object to it.

Without the ChartArea object, the Chart object appears only as a selectable area. This is because the Chart component requires at least one ChartArea object to be added to it. As chart areas and their container (Chart Area Collection) are complex data types, the Property Window editor in the iFIX WorkSpace cannot handle them. The action, however, can be done with VBA scripting, as shown in the following example.

IMPORTANT: This example assumes that you have the Microsoft Northwind sample database for SQL Server or SQL Server Express for this example to work. If you use SQL Server, the Northwind database may already be installed. To verify, check that Northwind is one of the databases installed on your system. If it is not installed (for either SQL Server or SQL Server Express), you can obtain it from the Microsoft web site.

1. In the iFIX WorkSpace, create a new picture.
2. In the iFIX WorkSpace, in Ribbon view, click the Insert tab.
3. On the Insert tab, select Objects/Links, and then click .NET Component. The .NET Component Browser dialog box appears.
4. In the tree browser, in the .NET Components folder, open .NET Framework, Windows Forms, System.Windows.Forms.DataVisualization, and then select the Chart component and click OK.
5. Optionally, resize the object. Be aware that in the iFIX WorkSpace, you should be able to see the object handles when you click on it, but the chart object will be transparent in Configure mode; this behavior is expected.
6. In the iFIX WorkSpace, right-click the Chart component and select Edit Script. The Microsoft Visual Basic Editor appears again.
7. On the Tools menu, click References. The References dialog box appears.
8. Scroll through the list of components and make sure check marks appear next to the following references, and click OK:
   - GEIP_Orion_DataConversion
   - Microsoft ActiveX Data Objects Library (version 2.7 or higher)
9. In the global area, under (General), add the following code to the global code section:

   ```
   Dim chartArea As GEIP_Orion_DataConversion.FormsChartArea
   Dim mouseDown As Boolean
   ```

   The first line defines a variable for the Chart Area object, and second line is for the component event handlers for the examples in the Using Event Handlers section.

10. Click the Save button to save the script.
11. Select the CFixPicture object from the object combo box on the top-left of the scripting window. Ignore or delete the KeyDown handler code.
12. Select the Initialize handler of the picture object from the top-right event combo box, and an Initialize handler will be added to the script.
13. Replace the CFixPicture_Initialize code with the following code snippet:

```
Private Sub CFixPicture_Initialize()
   Dim conn As New Connection
   Dim rs As New Recordset

   Set chartArea = Chart1.ChartAreas.Add("area1")
   chartArea.BackColor = RGB(0, 254, 1)
   chartArea.BackGradientStyle = GradientStyle_DiagonalRight
   chartArea.Area3DStyle.Enable3D = True

   conn.ConnectionString = "Provider=SQLNCLI10.1;Data Source=localhost;" _
       & "Initial Catalog=Northwind;Integrated Security=SSPI;"
    conn.Open

    rs.Open "Select Top 12 CustomerID, Freight from Orders", conn
    Chart1.DataBindTable rs, "CustomerID"
    rs.Close

    Chart1.Series(0).ChartType = SeriesChartType_Column
    Chart1.Series(0).SetCustomProperty "DrawingStyle", "Cylinder"
   conn.Close

End Sub
```

NOTE: You may need to modify the Provider or Data Source settings to specify different database settings. The code above assumes you are using the Northwind database on your local computer. For instance, if you are using SQL Server on another computer, and the computer name where the server resides is MyComputer, you would replace .\localhost with MyComputer.

In the second code paragraph of this code snippet, the chartArea object is set to the result of calling the Chart Area Collection object's method: Add(). The collection object is a property of the Chart1 object, which has been added when the Chart component is constructed.

NOTE: For different system configurations (SQL Server, database, and tables), the code for the connection string and record set selection and setting will require appropriate changes. The second string parameter of the DataBindTable method must be a string column of the selected table, and the other selected columns must be numeric. The other valid values for the second string parameter of the SetCustomProperty method are Emboss, LightToDark, Wedge, and Default (refer to Microsoft help on System.Windows.Forms.DataVisualization).

14. Click the Save button on the toolbar to save the script.
15. Close the Microsoft Visual Basic Editor, and save the picture in the iFIX WorkSpace.
16. Select Run to view your picture. The chart should display similar to the following figure. (This graph, however, was resized to a longer shape, after it was placed into the WorkSpace in step 5.)

Code from the Example

When you are finished with the example, your code should look similar to this:

```
Dim chartArea As GEIP_Orion_DataConversion.FormsChartArea
Dim mouseDown As Boolean

Private Sub CFixPicture_Initialize()
    Dim conn As New Connection
    Dim rs As New Recordset

    Set chartArea = Chart1.ChartAreas.Add("area1")
    chartArea.BackColor = RGB(0, 254, 1)
    chartArea.BackGradientStyle = GradientStyle_DiagonalRight
    chartArea.Area3DStyle.Enable3D = True
    conn.ConnectionString = "Provider=SQLNCLI10.1;Data Source=localhost;" _
        & "Initial Catalog=Northwind;Integrated Security=SSPI;"
    conn.Open

    rs.Open "Select Top 12 CustomerID, Freight from Orders", conn
    Chart1.DataBindTable rs, "CustomerID"
    rs.Close

    Chart1.Series(0).ChartType = SeriesChartType_Column
    Chart1.Series(0).SetCustomProperty "DrawingStyle", "Cylinder"
    conn.Close
End Sub

Private Sub Chart1_MouseDown(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    mouseDown = True
End Sub

Private Sub Chart1_MouseMove(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    If mouseDown = True Then
        chartArea.Area3DStyle.Inclination = (x Mod 90) / 2
        chartArea.Area3DStyle.Rotation = (y Mod 180) / 2
      End If
End Sub

Private Sub Chart1_MouseUp(ByVal state As Long, ByVal x As Long, ByVal y As Long)
   mouseDown = False
End Sub
```

Using Event Handlers

As the generic wrapper exposes all the events defined by an underlying .NET component as a COM connection point interface, the component events can be handled with VBA scripting.

IMPORTANT: These examples assume that you have the Microsoft Northwind sample database for SQL Server or SQL Server Express for this example to work. If you use SQL Server, the Northwind database may already be installed. To verify, check that Northwind is one of the databases installed on your system. If it is not installed (for either SQL Server or SQL Server Express), you can obtain it from the Microsoft web site.

Example 1

This first example takes the Chart component from the Windows Forms (in the .NET Component Browser dialog box: .NET Components > .NET Framework > Windows Forms > System.Windows.Forms.DataVisualization > Chart), and adds the MouseDown, MouseMove, and MouseUp event handlers.

1. Open the example iFIX picture from the Accessing Component Properties and Methods Through Scripting section, and save it under a new name.
2. In the iFIX WorkSpace, right-click the Chart component and select Edit Script. The Microsoft Visual Basic Editor appears.
3. Select the Chart1 object from the top-left object combo box, and then select the MouseDown event. An empty event handler will be inserted to the current script.
4. Add a simple line to set the MouseDown variable to True. For example, the finished handler will look similar to the following:

```
Private Sub Chart1_MouseDown(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    mouseDown = True
End Sub
```

5. Select the MouseMove event to add the event handler, and replace the code that was automatically added with the following:

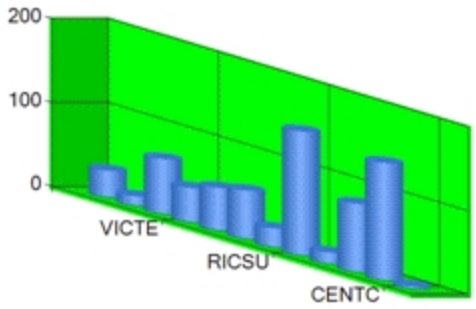
```
Private Sub Chart1_MouseMove(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    If mouseDown = True Then
        chartArea.Area3DStyle.Inclination = (x Mod 90) / 2
        chartArea.Area3DStyle.Rotation = (y Mod 180) / 2
    End If
End Sub
```

6. Select the MouseUp event to add the event handler, and replace the code that was automatically added with the following:

```
Private Sub Chart1_MouseUp(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    mouseDown = False
End Sub
```

   NOTE: The original .NET events have been converted to Win32-like messages in this example. The X and Y values are the X and Y coordinates of the mouse pointer, respectively. The state value is the mouse button state (not used here).

7. Click the Save button on the toolbar to save the script.
8. Go back to the iFIX WorkSpace, and switch to the Run mode.
9. Click the left mouse button on the chart graph, hold the button, and move the mouse down. The chart will rotate and incline according the current mouse position. The following figure shows an example of the chart at a rotated and inclined position.



NOTE: The logic in the MouseMove handler above is simplified to emphasize the main points of event handling. More sophisticated code is needed to make the graph move more smoothly in rotation and inclination.

Code from Example 1

When you are finished with Example 1, your code should look similar to this:

```
Dim chartArea As GEIP_Orion_DataConversion.FormsChartArea
Dim mouseDown As Boolean

Private Sub CFixPicture_Initialize()
    Dim conn As New Connection
    Dim rs As New Recordset

    Set chartArea = Chart1.ChartAreas.Add("area1")
    chartArea.BackColor = RGB(0, 254, 1)
    chartArea.BackGradientStyle = GradientStyle_DiagonalRight
    chartArea.Area3DStyle.Enable3D = True
    conn.ConnectionString = "Provider=SQLNCLI10.1;Data Source=localhost;" _
        & "Initial Catalog=Northwind;Integrated Security=SSPI;"
    conn.Open

    rs.Open "Select Top 12 CustomerID, Freight from Orders", conn
    Chart1.DataBindTable rs, "CustomerID"
    rs.Close

    Chart1.Series(0).ChartType = SeriesChartType_Column
    Chart1.Series(0).SetCustomProperty "DrawingStyle", "Cylinder"
    conn.Close
End Sub

Private Sub Chart1_MouseDown(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    mouseDown = True
End Sub

Private Sub Chart1_MouseMove(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    If mouseDown = True Then
        chartArea.Area3DStyle.Inclination = (x Mod 90) / 2
        chartArea.Area3DStyle.Rotation = (y Mod 180) / 2
      End If
End Sub

Private Sub Chart1_MouseUp(ByVal state As Long, ByVal x As Long, ByVal y As Long)
   mouseDown = False
End Sub
```

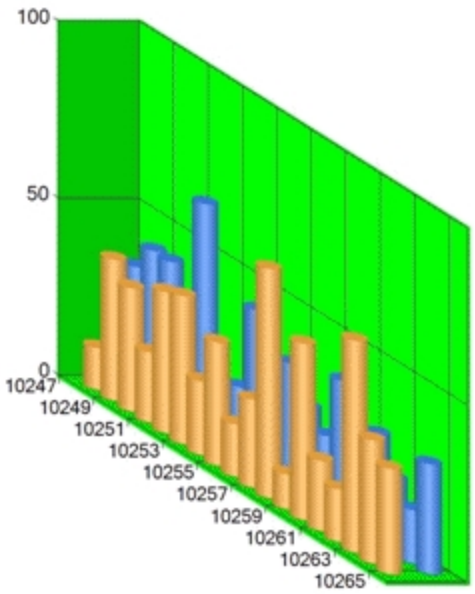
Example 2

This second example takes the same Chart component from the Windows Forms (in the .NET Component Browser dialog box: .NET Components > .NET Framework > Windows Forms > System.Windows.Forms.DataVisualization > Chart), with the MouseDown, MouseMove, and MouseUp events, and adds code for two columns. These columns come from another table in the Northwind database. The code between the conn.Open and conn.Close references in the CFixPicture_Initialize() handler from the first example is replaced with new code.

1. Open the example iFIX picture from the example above. Save it under a new name.
2. In the iFIX WorkSpace, right-click the Chart component and select Edit Script. The Microsoft Visual Basic Editor appears.
3. Replace the code between conn.Open and conn.Close in the CFixPicture_Initialize() handler with the following code (this code selects two numeric columns from another table in the Northwind database):

```
rs.Open "select Top 50 OrderID, UnitPrice, Quantity from [Order Details]", conn
Chart1.DataBindTable rs, "OrderID"
rs.Close

Chart1.Series(0).ChartType = SeriesChartType_Column
Chart1.Series(0).SetCustomProperty "DrawingStyle", "Cylinder"
Chart1.Series(1).ChartType = SeriesChartType_Column
Chart1.Series(1).SetCustomProperty "DrawingStyle", "Cylinder"
```

4. Click the Save button on the toolbar to save the script.
5. Go back to the iFIX WorkSpace, and switch to the Run mode. The following figure shows an example of the chart, with two numeric columns and rotated.



Code from Example 2

When you are finished with Example 2, your code should look similar to this:

```
Dim chartArea As GEIP_Orion_DataConversion.FormsChartArea
```

```
Dim mouseDown As Boolean

Private Sub CFixPicture_Initialize()
   Dim conn As New Connection
   Dim rs As New Recordset
   Set chartArea = Chart1.ChartAreas.Add("area1")
   chartArea.BackColor = RGB(0, 254, 1)
   chartArea.BackGradientStyle = GradientStyle_DiagonalRight
   chartArea.Area3DStyle.Enable3D = True
    conn.ConnectionString = "Provider=SQLNCLI10.1;Data Source=localhost;" _
       & "Initial Catalog=Northwind;Integrated Security=SSPI;"
    conn.Open

        rs.Open "select Top 50 OrderID, UnitPrice, Quantity from [Order Details]", conn
    Chart1.DataBindTable rs, "OrderID"
    rs.Close

    Chart1.Series(0).ChartType = SeriesChartType_Column
    Chart1.Series(0).SetCustomProperty "DrawingStyle", "Cylinder"
    Chart1.Series(1).ChartType = SeriesChartType_Column
    Chart1.Series(1).SetCustomProperty "DrawingStyle", "Cylinder"

   conn.Close
End Sub

Private Sub Chart1_MouseDown(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    mouseDown = True
End Sub

Private Sub Chart1_MouseMove(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    If mouseDown = True Then
        chartArea.Area3DStyle.Inclination = (x Mod 90) / 2
        chartArea.Area3DStyle.Rotation = (y Mod 180) / 2
    End If
End Sub

Private Sub Chart1_MouseUp(ByVal state As Long, ByVal x As Long, ByVal y As Long)
    mouseDown = False
End Sub
```

Handling Events with Non-Converted Parameters

The event handlers, such as the MouseDown, MouseMove, and MouseUp event handlers described in the Using Event Handlers section, are converted by the data conversion module from the corresponding .NET event handlers. For component hosting in iFIX, key event handlers are converted to names similar to their counterpart Win32 key messages.

Any non-converted parameters defined by a component are represented as Win32-like parameters, but with two string arguments: the name of the event raiser, and the .NET argument type.

These string arguments are not very useful in these non-converted (more correctly, not fully converted) cases. However, as the events are actually raised in the meaningful manners, these events can be handled in a normal manner using the event raiser's properties and methods to access companion information.

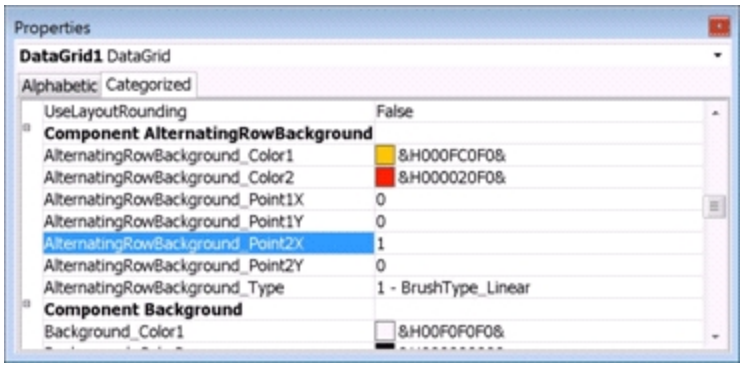
Example

This example takes the DataGrid and ComboBox components from the Windows Presentation Framework (in the .NET Component Browser dialog box: .NET Components > .NET Framework > Windows Presentation Framework > DataGrid, and ComboBox), and adds some code to alternate the row color in the table, and to allow you to use the ComboBox control to select a row in the DataGrid.

First, with the DataGrid object, we configure the AlternatingRowBackground component property. Then, for the ComboBox object, we add the SelectionChanged event with two string parameters, because the Windows Presentation Framework SelectionChangedEventArgs class contains unconverted parameters. Both the DataGrid and ComboBox components support the SelectionIndex property, which allows you to select a corresponding row in the DataGrid, using the ComboBox control.

IMPORTANT: This example assumes that you have the Microsoft Northwind sample database for SQL Server or SQL Server Express for this example to work. If you use SQL Server, the Northwind database may already be installed. To verify, check that Northwind is one of the databases installed on your system. If it is not installed (for either SQL Server or SQL Server Express), you can obtain it from the Microsoft web site.

1. In the iFIX WorkSpace, create a new picture.
2. In the iFIX WorkSpace, in Ribbon view, click the Insert tab.
3. On the Insert tab, select Objects/Links, and then click .NET Component. The .NET Component Browser dialog box appears.
4. In the tree browser, in the .NET Components folder, open .NET Framework, Windows Presentation Framework, Presentation Framework, and then select the DataGrid component and click OK.
5. On the Insert tab, select Objects/Links, and then click .NET Component. The .NET Component Browser dialog box appears.
6. In the tree browser, in the .NET Components folder, open .NET Framework, Windows Presentation Framework, Presentation Framework, and then select the ComboBox component and click OK. Insert the ComboBox below the DataGrid in the iFIX picture.
7. In the iFIX WorkSpace, right-click the DataGrid component and select Property Window, if it is not already displayed. The Property Window appears in the iFIX WorkSpace.
8. Click the Categorized tab, and scroll to the Component AlternatingRowBackground category for the DataGrid object as shown in the following figure. Change the property settings to match the AlternatingRowBackground_Color1, AlternatingRowBackground_Color2, AlternatingRowBackground_Point2X, and AlternatingRowBackground_Type properties in the following figure:



When you are complete, the following four properties should have these settings:

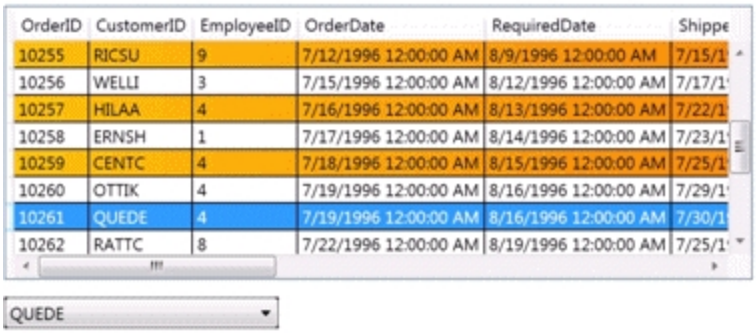
- AlternatingRowBackground_Color1 as &H000FF0F0&
- AlternatingRowBackground_Color2 as &H000020F0&
- AlternatingRowBackground_Point2X as 1
- AlternatingRowBackground_Type as 1 - BrushType_Linear.

9. Save the picture.
10. In the iFIX WorkSpace, right-click the DataGrid component and select Edit Script. The Microsoft Visual Basic Editor appears.
11. On the Tools menu, click References. The References dialog box appears.
12. Scroll through the list of components and make sure check marks appear next to the following references:
    - Microsoft ActiveX Data Objects Library (version 2.7 or higher)
    - GEIP_Orion_DataConversion
13. Select the Initialize handler of the picture object from the top-right event combo box, and an Initialize handler will be added to the script. Replace the initialize handler with the following code:

```
Private Sub CFixPicture_Initialize()
    Dim conn As New Connection
    Dim rs As New Recordset

    conn.ConnectionString = "Provider=SQLNCLI10.1;Data Source=.\sqlexpress;" _
        & "Initial Catalog=Northwind;Integrated Security=SSPI;"
    conn.Open

    rs.Open "Select Top 20 * from Orders", conn
    Set DataGrid1.ItemsSource = rs

    rs.MoveFirst
    Do While Not rs.EOF
        ComboBox1.Items.Add CStr(rs!CustomerID)
        rs.MoveNext
    Loop
    ComboBox1.SelectedIndex = 0
    rs.Close

    conn.Close
End Sub
```

14. Click the Save button on the toolbar to save the script.
15. Go back to the iFIX WorkSpace, and switch to the Run mode. The two objects will look similar to those in the following figure.

    NOTE: These example objects were resized after they were placed in the picture. You can optionally resize your objects too. Be aware that in the iFIX WorkSpace, you should be able to see the object handles when you click on the grid, but the grid object will be transparent in Configure mode; this behavior is expected.



16. Right-click the ComboBox object and select, Edit Script.
17. Add the SelectionChanged event to the ComboBox object. An event handler will be inserted (but with two string parameters because the WPF SelectionChangedEventArgs class is not converted).
18. Change the event handler code so that it matches the code below, as demonstrated by the following code snippet:

```
Private Sub ComboBox1_SelectionChanged(ByVal addedItems As Variant, ByVal removedItems As Variant)
    DataGrid1.SelectedIndex = ComboBox1.SelectedIndex
    DataGrid1.ScrollIntoView DataGrid1.SelectedItem
End Sub
```

    As the DataGrid component contains a method that supports scrolling of the view port, the line of code referring to the ScrollIntoView method makes the selected row always visible in this example. With this event scripting, the ComboBox object can now drive the DataGrid object.

19. Save the picture, and switch to run mode. Try using the drop-down list to change the row focus in the grid.

    The selection of an item in the ComboBox is reflected accordingly, as a selection of a row in the DataGrid.

Code from the Example

When you are finished with the example, your code should look similar to this:

```
Private Sub CFixPicture_Initialize()
    Dim conn As New Connection
    Dim rs As New Recordset
    conn.ConnectionString = "Provider=SQLNCLI10.1;Data Source=localhost;" _
        & "Initial Catalog=Northwind;Integrated Security=SSPI;"
    conn.Open
    rs.Open "Select Top 20 * from Orders", conn
    Set DataGrid1.ItemsSource = rs
   rs.MoveFirst
    Do While Not rs.EOF
        ComboBox1.Items.Add CStr(rs!CustomerID)
       rs.MoveNext
   Loop
   ComboBox1.SelectedIndex = 0
   rs.Close
   conn.Close
End Sub

Private Sub ComboBox1_DropDownOpened(ByVal param1 As String, ByVal param2 As String)
End Sub
```

```
Private Sub ComboBox1_SelectionChanged(ByVal addedItems As Variant, ByVal removedItems As Variant)
    DataGrid1.SelectedIndex = ComboBox1.SelectedIndex
    DataGrid1.ScrollIntoView DataGrid1.SelectedItem
End Sub

Private Sub DataGrid1_ColumnDisplayIndexChanged(ByVal param1 As String, ByVal param2 As String)
End Sub
```

NOTE: The SelectionChanged event is actually converted with two parameters of Variant(SafeArray). The SafeArray interface requires more advanced scripting and is not used here.

Using Properties and Methods of the iFIX Container

The hosting environment and the iFIX component container work together to combine the properties and methods of the container with those of the underlying .NET control. This makes it possible to access the properties of the container and the .NET control from within the iFIX Properties Window or to access the properties and methods with a single variable inserted by the VBA scripting tool.

The following common properties of the component container are grouped under the label Misc, in the iFIX Properties Window for the .NET Component:

- Name
- Cancel
- ContextID
- ControlOrderIndex
- Default
- Description
- EnableTooltips
- Height
- HighlightEnabled
- HorizontalPosition
- HorizontalScaleDirection
- HorizontalScalePercentage
- IsSelectable
- Layer
- UniformScale
- VerticalPosition
- VerticalScaleDirection
- VerticalScalePercentage
- Visible
- Width

For more information on the meaning and usage of these properties and additional properties and methods available with VBA scripting, refer to the iFIX Automation Reference.

Advanced Features

The hosting environment supports hosting and interfacing with the built-in Windows Presentation Framework or Windows Forms (WinForms) controls. Advanced features are provided to allow you to build components from any .NET controls - your own or from a third party - created from these built-in controls. The following sections provide more information on how to use this advanced functionality:

- Creating New Components
- Using New Components on Other iFIX Systems
- Data Conversion Rules
- Error Logging
- Sample Projects in Visual Studio

Creating New Components

In addition to the build-in Windows Presentation Framework or Windows Forms controls, you can use third-party controls or create your own controls based on any built-in controls. For more information on the task you want to perform, refer to the following sections:

- Creating a .NET Control
- Deleting Nodes on the Component Browser
- Supporting Files for .NET Components
- Uniqueness of .NET Control Assembly Names

Creating a .NET Control

In addition to using a third-party control, you can also create your own custom .NET component, as the example below describes. Any custom or third-party .DLL file should reside in the *<iFIX install>*\DotNet Components folder.

Custom or third-party controls can be derived and hosted from the following pre-built controls, following the .NET Framework 4.0 classes:

- For Windows Forms: System.Windows.Forms.Control and any of approximately 60 control classes in the System.Windows.Forms namespace.
- For Windows Presentation Framework: System.Windows.UIElement, System.Windows.FrameworkElement, and any of the approximately 100 control classes in the System.Windows.Controls namespace.

NOTE: All custom controls must define a parameter-less constructor or no constructors at all (so that the compiler will add a parameter-less one). This step is required so that custom controls can be created in an environment that requires no parameters to be passed in. All ActiveX controls working with a COM client and .NET controls used by a user-interface configurable .NET environment require this configuration.

You can add the new components to your .NET Component Browser dialog box, as described in the Adding Components to the .NET Component Browser Dialog Box section.

Additionally, you can then copy the associated .DLL files to each iFIX install where you want to display the picture, as well as the GEIP.Orion.Components.dat file, as described in the Copying Component files to Other Systems section.

Example

The following example shows how to create a Trend Chart within the .NET Framework 4.0.

1. Create a folder under *<iFIX install>*\DotNet Components named: My Sample Components.
2. Use the Notepad text editor to save the following code as a C# source file named TrendChart.cs.

```
using System;
using System.Drawing;
using System.Windows.Forms.DataVisualization.Charting;
namespace ChartControls
{
    public class TrendChart : Chart
    {
        private double _value;
        private ChartArea _area;
        private Series _series;
        private Title _title;
        public TrendChart()
        {
            BackColor = Color.Silver;
            BorderSkin.SkinStyle = BorderSkinStyle.FrameThin5;
            _title = Titles.Add("Trend Chart");
            _title.Font = new Font("Arial", 10, FontStyle.Bold);
            _area = ChartAreas.Add("area0");
            _area.AxisX.LabelStyle.Format = "hh:mm:ss";
            _area.AxisX.LabelStyle.Interval = 5;
            _area.AxisX.LabelStyle.IntervalType = DateTimeIntervalType.Seconds;
            _area.AxisX.MajorGrid.Interval = 5;
            _area.AxisX.MajorGrid.IntervalType = DateTimeIntervalType.Seconds;
            _area.BackColor = Color.Orange;
            _area.BackGradientStyle = GradientStyle.TopBottom;
            _series = Series.Add("series0");
            _series.ChartType = SeriesChartType.Line;
            _series.ShadowOffset = 1;
        }
        public string Title
        {
            get { return _title.Text; }
            set { _title.Text = value; }
        }
        public double Value
        {
            get { return _value; }
            set
            {
                _value = value;
                DateTime current = DateTime.Now;
                _series.Points.AddXY(current.ToOADate(), _value);
                double removeBefore = current.AddSeconds(-25).ToOADate();
                while (_series.Points[0].XValue < removeBefore)
```

```
                    _series.Points.RemoveAt(0);
                _area.AxisX.Minimum = _series.Points[0].XValue;
                _area.AxisX.Maximum
                    = DateTime.FromOADate(_series.Points[0].XValue).AddSeconds(30).ToOADate();
            }
        }
    }
}
```

NOTE: The TrendChart class in this above example is derived from the Framework 4.0 Chart class. In the constructor, a ChartArea and a Series are added to the chart control, which is then configured by setting some ChartArea and Series properties. Two properties are defined: Title for setting a chart title, and Value for binding to a data source. When the data value changes, the set action of the Value property adds a new data point to the chart. If the number of data points excesses a preset limit, the oldest points are removed.

3. Save this text file as TrendChart.cs, and copy it into the My Sample Components folder created in step 1.
4. Open a Command Prompt window, change the directory to the *<iFIX install>*\DotNet Components\My Sample Components folder, and issue the following command:

   C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\csc /t:library /out:ChartControls.dll /r:System.dll;System.Windows.Forms.dll;System.Windows.Forms.DataVisualization.dll TrendChart.cs

   This command assumes that .NET Framework 4.0 is installed in the default folder. Observe that the compiled assembly ChartControls.dll is saved into the My Sample Components folder.

   NOTE: The TrendChart control from this example can also be built with Visual Studio 2010. A sample project for this control is included in the folder <iFIX install>\DotNet Components\Sample Components\VS2010SampleProjects folder.
5. Add your new component to the .NET component Browser dialog box, so that you can add it to your iFIX picture. For steps, refer to the Adding Components to the .NET Component Browser Dialog Box section

Deleting Nodes on the Component Browser

The .NET Component Browser dialog box includes a button labeled Delete Node, to allow a user to delete a Group or Assembly node on the browser. This feature is necessary to keep the browser clean and well organized, but use care when deleting a component assembly node (such as the ChartControls node). An iFIX picture will fail to load if it contains any control objects created from the components of the deleted assembly. The deletion action is safeguarded by the following rules:

- The root Group node (.NET Components) may not be deleted.
- A Group node may not be deleted if it contains child nodes.
- An Assembly node may not be deleted if any opened iFIX pictures contain controls objects created from its components.

An issue could occur if a closed picture that contains components from the deleted assembly is then opened. In this case, because the iFIX WorkSpace does not check references in the closed pictures, an error may occur on load. This is the same behavior as that of an unregistered ActiveX control in the iFIX WorkSpace.

Supporting Files for .NET Components

Supporting files for .NET components are automatically created from the original .NET source assembly (such as the ChartControls.dll) when you add the assembly through the .NET Component Browser dialog box. Assemblies referenced by the source assembly and not installed in the assembly cache must be put in the same folder. If you installed to the default location, this folder is: C:\Program Files (x86)\GE\iFIX\DotNet Components.

The component creation process creates the following essential files for a .NET source assembly to support hosting and communicating with all the components created from the selected controls in the source assembly:

- *<Occ><ordinal number>_<source assembly name>*.dll – the component assembly.
- *<Occ><the same ordinal number>_<source assembly name>*.tlb – the component type library.

Some of the temporary or debugging files may be created when you create a .NET component, but they are not essential for the component to function. If the same source assembly is selected to the Add Components process the second time, the ordinal number will increase.

NOTES:

- Do not manually delete the essential supporting files.
- Occ is short for Orion COM Component.

Uniqueness of .NET Control Assembly Names

The original control assembly name (an example of a control assembly name is ChartControls.dll), and any other control assembly names associated with it, must be unique among all .NET control assemblies that are used to build iFIX hostable components. Otherwise, unexpected results may happen as the .NET runtime will use the first loaded control assembly for all same named assemblies.

NOTE: .NET has provisions to support the loading assemblies with the same name but different versions. However, that requires the versioned assemblies be signed and installed into the system assembly cache, which is not a supported scenario for GE component hosting.

.NET Component Directories

All original .NET control assemblies must be stored under the root component directory: <iFIX Install>\.NET components. When an original control assembly is selected to build iFIX hostable components, the supporting assembly and type library are saved in the same directory. When components are copied from one system to another, the folder hierarchy must be maintained.

Network shared folders for component storage are not supported due to .NET security implementations.

Using New Components on iFIX Systems

After you create your new .NET components, if you want to use them on iFIX systems you need to:

1. Copy the compiled .NET Component files to each iFIX node that you want to use the new controls.

2. Add the new components to the .NET Component Browser dialog box on each iFIX node where you want to display them.

For steps, refer to the sections above.

Copying Compiled Component Files to iFIX Nodes

If new .NET Components are created and inserted into iFIX pictures on one iFIX system, and the saved pictures are opened on another iFIX system, the pictures will not load with the new components if the corresponding component files do not exist on the new system. This is true for ActiveX controls that are not installed (copied and registered) on a new target system.

There is no need to register the new components; you do not need to register the component assembly or the component type library files. You only need to copy the associated files to other systems, using the same directory hierarchy, relative to the iFIX install folder. For example, if you modified the TrendChart control, you would copy the ChartControls.dll, Occ.ChartControls.dll, and Occ.ChartControls.tlb to the *<iFIX install>*\DotNet Components\Sample Components folder of the other iFIX system (even if it is a client with mapped drives in use). The TrendChart objects configured on the original system will then load and function on the new system.

Any custom or third-party supporting assemblies in the iFIX install folder must also be copied to the iFIX install folders of other systems. Some third-party common assemblies may need to be installed into the system assembly cache or be copied to the iFIX install folder.

Basically, the *<iFIX install>*\DotNet Components folder should be the same on each iFIX node.

Adding Components to the .NET Component Browser Dialog Box

After you create a new custom .NET component, you need to add it to the tree view area in the .NET Component Browser dialog box in order to add the control to an iFIX picture on your system. Use the interface in the Component Browser dialog box to add a new component to an individual system.

Later, if you want to synchronize this new component across all of your iFIX system (clients and servers), manually copy the all of the files in the DotNet Components folder, including subfolders, and the master browser file (GEIP.Orion.Components.dat file) and any other .dll files your assemblies use, to each system that you want to synchronize. Basically, the *<iFIX install>*\DotNet Components should be the same on each iFIX node.

▶ To add a new component to a single system using the Component Browser dialog box:

1. In the iFIX Workspace, in Ribbon view, click the Insert tab.
2. On the Insert tab, select Objects/Links, and then click .NET Component. The .NET Component Browser dialog box appears.
3. With the top level folder selected, .NET Component, click the Add Group button. The Add a Component Group dialog box appears.
4. Enter a group name and description, and click OK. (The Description field is optional, but if text is entered, it will display as a tooltip when the mouse hovers on the group node.)

5. With the new group selected, click the Add Components button. The Add .NET Components dialog box appears.
6. Click the browse (...) button to select a component source. The dialog box populates with the Component Assembly Name, along with the associated classes.
7. Select the check box next to each component class that you want to add.
8. Click OK. The entries should now appear in the browser.

To synchronize components across all systems:

1. On the computer that contains the master copy, make a copy of the *<iFIX install>*\DotNet Components folder, and the *<iFIX install>*\GEIP.Orion.Components.dat file.
2. On each system that you want to synchronize, follow these steps:
    a. Copy the master DotNet Components folder over the original *<iFIX install>*\DotNet Components folder on the system you want to synchronize. Copy over this entire folder (including all files and subfolders).
    c. Copy the master GEIP.Orion.Components.dat file, the Component Browser content file, over the original *<iFIX install>*\GEIP.Orion.Components.dat file on the system you want to synchronize.

Copying Pictures That Include .NET Components

You can simply move (copy) pictures that include .NET components from one machine to another if the .NET components in the picture are all listed on the Component Browser and the supporting assemblies are all in the same hierarchical folders of the current machine. Otherwise, you must copy the supporting assemblies and add .NET controls to the browser. If the .NET components on the other machine are a superset of those on the current machine, you may simply copy the DotNet Components folder and the GEIP.Orion.Components.dat file.

Re-linking Components from Another iFIX Machine

If a custom .NET component exists on another iFIX machine and you want to re-link it to your current machine, you can use the .NET Component Browser to add the component to the tree view area in the .NET Component Browser dialog box. You will then be able to add the component to iFIX pictures on your system.

To re-link a custom component from another iFIX machine:

1. On the computer that contains the custom component, make a copy of the .NET Component Assembly folder.
2. Browse to the DotNet Components folder of the computer to which you want to re-link the component, and move the copied folder there.
3. Start iFIX and launch WorkSpace on the computer to which you want to re-link the component.
4. On the Insert tab, select Object/Links, and then click .NET Component. The component Browser appears.

5. Select the group for the component and then click Add Components. The Add .NET Components dialogue appears.
6. Click the browse button (...) and browse to the .NET component folder that was copied and click the Occ.<filename>.dll.

   IMPORTANT: It is essential at this point that you do not select the original source assembly (*FileName*.dll without the Occ. prefix). Otherwise, another version of the component assembly will be created as Occ1.*FileName*.dll, which is not needed for this process. The goal is only to re-link the component assembly (Occ.*FileName*.dll) that has already been created on the other iFIX machine.
7. Give the component assembly a new name or leave the current name. Select the components to be added and then click OK. Verify that the component displays in the .NET Component browser, select it, and click OK.

Data Conversion Rules

The data types that iFIX understands are different from the .NET data types. The .NET Framework provides data conversions only for primitive data types, such as integers, floats and strings. The hosting environment in iFIX supports converting commonly used complex data types:

- .NET DataTable, DataSet, IEnumerable to/from COM ADODB.Recordset.
- .NET arrays of primitive types to/from COM SAFEARRAY.
- .NET event handlers are converted to Windows message-style handlers.
- Some .NET collection types to/from specific COM custom types defined in this module.

Properties and Methods

If a custom or third-party .NET control is one of the types listed in the Supported .NET Control Types section, any public property or method defined in the control class will be exposed. This assumes that the data type of the property, or the type of each parameter in the method, is one of the following types:

- Primitive types, including the integral and floating numeric types.
- Microsoft Component Object Model recognizable types, including the String, DateTime and System.Drawing.Color (WinForms Color) types.
- COM visible types, such as System.Collections.ArrayList, System.Collections.IList and System.Object.
- NET generic list types: List<short>, List<int>, List<float>, List<double> and List<string> (all of the System.Collections.Generic namespace).
- The DataTable and DataView classes (all of the System.Data namespace), the System.Windows.Media.Brush class, the System.Collections.IEnamerable interface, and the System.Windows.Control.ItemCollection class.
- The Windows Forms Font (System.Drawing.Font) and WPF FontFamily (System.Windows.Media.FontFamily) classes and the FontStretch, FontStyle and FontWeight structures (all of the System.Windows namespace).

The property or method in the exposed COM interface will have the corresponding COM types converted from the original .NET types.

If a property or method does not meet the above conditions, it will be ignored in the component building process and thus not be available in the component COM interface.

Events

All the public events defined in the control class are exposed as COM connection point handlers, according to the following rules:

- If all the parameter types of a .NET event delegate are convertible according to the conditions listed in the Properties and Methods section above, the .NET event delegate will be converted to a COM event handler with the same number of parameters of the corresponding COM types.
- Some common .NET delegate types are converted to specifically defined COM event handlers.

  These include the mouse events:

  - For Windows Presentation Framework: MouseEventHandler, MouseButtonEventHandler, and MouseWheelEventHandler.
  - For Windows Forms: MouseEventHandler delegates.

  The key events:

  - For Windows Presentation Framework: KeyEventHandler, and KeyboardEventHandler.
  - For Windows Forms: KeyEventHandler, and KeyPressEventHandler delegates.

  The selection events:

  - SelectionChangedEventHandler and EventHandler<SelectionChangedEventArgs> delegates.
- The converted COM mouse event handlers are defined to have three parameters: the mouse button state, and the X and Y positions. The key COM event handlers are defined to have two parameters: the key state and the key code. For more information on the mouse button state and the key state, refer to the Microsoft Win32 API help.
- The converted COM selection event handler has two parameters of the COM Variant type with an array subtype: addedItems for items newly added to, and removedItems for items newly removed from the selected items collection. To avoid array indexing problems (the items arrays might be empty), check array bounds with the UBound VBA function.
- All other delegates that follow the .NET convention (two parameters: sender of the Object type and of an EventArgs derived type) are converted to a COM handler of two parameters of the String type.

Passing Complex Data Types

The .NET Component hosting environment supports passing complex data types such as arrays (lists) and data tables. The supported .NET types are exposed as common COM types or specific interfaces and types implemented by the conversion module. The supported COM types may be passed to or returned from the methods and properties of .NET controls. As a result of data conversion, these COM data types appear as the corresponding .NET types to the NET controls.

DataTable, DataView (both of System.Data), and IEnumerable (of System.Collection)

These data types appear as ADODB Recordsets and are used to pass data tables with columns and rows. IEnumerable is converted to Recordset because many .NET controls use the IEnumerable interface to pass DataTable objects. Be sure to add the Microsoft ActiveX Data Objects Library (2.7 or higher) reference when scripting with these date types. For more information on the usage of the Recordset interface, refer to the Microsoft help on ADODB.

List<short>, List<int>, List<float>, List<double>, and List<string> (all of System.Collections.Generic)

These data types appear as ShortList, IntList, FloatList, DoubleList, and StringList implemented in the data conversion module. Be sure to add the GEIP_Orion_Dataconversion reference when scripting with these date types. For example, the ShortList list implements the following interface:

```
interface IShortList : IDispatch
{
    [id(0x60020000), propget]
    HRESULT Count([out, retval] long* pRetVal);
    [id(0x60020001)]
    HRESULT Add([in] short item);
    [id(0x60020002)]
    HRESULT Clear();
    [id(00000000), propget]
    HRESULT item(
                    [in] long index,
                    [out, retval] short* pRetVal);
    [id(00000000), propput]
    HRESULT item(
                    [in] long index,
                    [in] short pRetVal);
    [id(0x60020005)]
    HRESULT Insert(
                    [in] long index,
                    [in] short item);
    [id(0x60020006)]
    HRESULT RemoveAt([in] long index);
};
```

The other lists implement basically the same interface, but with different data types (long for IntList, and so on) for the item parameter.

ArrayList (of System.Collections)

This data type appears as ObjectList, which implements the following interface:

```
interface IObjectList : IDispatch
{
    [id(0x60020000), propget]
    HRESULT Count([out, retval] long* pRetVal);
    [id(0x60020001)]
    HRESULT Add(
                    [in] VARIANT item,
                    [out, retval] long* pRetVal);
    [id(0x60020002)]
    HRESULT Clear();
    [id(00000000), propget]
    HRESULT item(
                    [in] long index,
                    [out, retval] VARIANT* pRetVal);
    [id(00000000), propputref]
    HRESULT item(
```

```
                    [in] long index,
                    [in] VARIANT pRetVal);
        [id(0x60020005)]
        HRESULT Insert(
                    [in] long index,
                    [in] VARIANT item);
        [id(0x60020006)]
        HRESULT RemoveAt([in] long index);
    };
```

As the item type listed above is VARIANT, this list can pass any convertible types supported by the data conversion module. For example, list objects can be added to an ObjectList as list items, resulting in a multi- dimensional array. Add the GEIP_Orion_Dataconversion reference when scripting with this data type.

Strategies for Parameter Types That Cannot Be Converted

iFIX pictures can not only host, but also communicate with the .NET controls through primitive data types and converted parameter types. The most commonly used parameter types are converted; however, there are parameter types, such as those with custom .NET controls, that cannot be converted. These unconverted parameter types are ignored.

In order to avoid the essential properties, methods, and events of a .NET control from being ignored (not exposed), you can do any of the following.

- Rewrite a desired property using the convertible types listed in the Data Conversion Rules section.
- Separate a desired property into multiple properties of the convertible types listed in the Data Conversion Rules section.
- Rewrite a method with the convertible types or with more parameters, so that each parameter can be converted.
- Define a new event type (delegate) with all convertible parameters, and reroute the original event with the new delegate.

Error Logging

The error logging module for the .NET Component is designed to log information and error messages to a file named ComponentsLogger.log. From this file, error causes can be analyzed. For instance, major errors such as a missing .dll file are logged to file. A error logged to this file might look similar to this:

```
2011-07-01 13:22:12,919 [1] ERROR GEIP.Orion.DotNetComAdapters.DotNetComAdapter
Failed to load the source assembly: C:\Program Files (x86)\GE\iFIX\DotNet Components\Sample Components\GaugeContr
```

By default, the ComponentsLogger.log file is saved to the following location: *<iFIX install>*\DotNet Components, on the computer where you are viewing the pictures. For instance, if you are viewing your pictures on an iClient (view node), the log file is saved on the iClient machine.

Default Logging Settings

The logging process is controlled by a logging configuration file named GEIP.Orion.ComponentsLogger.config in the iFIX install folder. By default, this file contains following content:

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net debug="true">
  <appender name="RollingLogFileAppender" type="log4net.Appender.RollingFileAppender">
    <file value=".\\DotNet Components\\ComponentsLogger.log" />
    <appendToFile value="true" />
    <rollingStyle value="Size" />
    <maxSizeRollBackups value="10" />
    <maximumFileSize value="10MB" />
    <staticLogFileName value="true" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%d [%t] %-5p %logger%n%m%n"/>
    </layout>
  </appender>
  <root>
    <level value="WARN" />
    <appender-ref ref="RollingLogFileAppender" />
  </root>
</log4net>
```

The items that can be most easily changed are the logging file path and the logging level. The default settings, as shown above, set the path to <iFIX install>\DotNet Components\ComponentsLogger.log and the logging level to WARN.

The logging levels are: DEBUG, INFO, WARN, ERROR, and FATAL. The level DEBUG logs everything, and the other levels: INFO, WARN, ERROR, and FATAL, log successively fewer messages. Be aware that the logging actions are mostly performed at the configuration mode, and there are no significant performance penalties in run mode even if the logging level is set to DEBUG.

To change the default logging level:

1. In Notepad, open the GEIP.Orion.ComponentsLogger.config file in the iFIX install folder.
2. Locate the following line, and change the level listed in quotes to the desired level:

   ```
   <level value="WARN" />
   ```

3. Restart iFIX.

Sample Projects in Visual Studio

iFIX installs the following Microsoft® Visual Studio® 2010 sample projects for the .NET Component:

- ChartControlsV2
- GaugeControlsV2

These projects are included in the *<iFIX install>*\DotNet Components\Sample Components\VS2010SampleProjects folder.

These samples are pure .NET controls targeted to a common .NET environment. The V2 suffix was added to avoid conflict with any code that you may have generated with previous code examples in this e-book.

The trend chart control, **ChartControlsV2**, is a Windows Forms (or WinForms) chart control. There are a few properties and a simple mouse event included.

The linear gauge control, **GaugeControlsV2**, is a Windows® Presentation Framework user control. There are a lot of properties added to make its overall look configurable, including title font and color, scale font and color, range limits and colors, scale mark sizes and colors, and so on. In run mode, the range color will blink if the needle enters a scaled region. There are also multiple events raised when the gauge needle crosses the limit borders. The needle may be dragged, and a new reading will be set and an event raised when the needle drops.

The control names and any other control assembly names must be unique among all .NET control assemblies that are used to build iFIX hostable components. Otherwise, unexpected results may occur as the .NET runtime will use the first loaded control assembly for all same named assemblies.

IMPORTANT: The full name (namespace and class name) of any .NET control must be unique.

The examples depend on only system assemblies (as listed for the /r command option, they are System.dll, System.Windows.Forms.dll and System.Windows.Forms.DataVisualization.dll). Third-party controls, however, usually reference their own supporting assemblies. When a control assembly is compiled, the compiler copies all non-system reference assemblies to the output folder.

The control assembly and its supporting reference assemblies should be copied to a specific folder under the <iFIX install>\DotNet Components folder. However, some third-party common assemblies may need to be installed into the system assembly cache or be copied to the iFIX install folder.

Index

H

I

K

L

M

N

O

P

R

S