Walkthrough: Hosting a Windows Forms Composite Control in WPF

.NET Framework 4.5 2 out of 2 rated this helpful

Windows Presentation Foundation (WPF) provides a rich environment for creating applications. However, when you have a substantial investment in Windows Forms code, it can be more effective to reuse at least some of that code in your WPF application rather than to rewrite it from scratch. The most common scenario is when you have existing Windows Forms controls. In some cases, you might not even have access to the source code for these controls. WPF provides a straightforward procedure for hosting such controls in a WPF application. For example, you can use WPF for most of your programming while hosting your specialized DataGridView controls.

This walkthrough steps you through an application that hosts a Windows Forms composite control to perform data entry in a WPF application. The composite control is packaged in a DLL. This general procedure can be extended to more complex applications and controls. This walkthrough is designed to be nearly identical in appearance and functionality to Walkthrough: Hosting a WPF Composite Control in Windows Forms. The primary difference is that the hosting scenario is reversed.

The walkthrough is divided into two sections. The first section briefly describes the implementation of the Windows Forms composite control. The second section discusses in detail how to host the composite control in a WPF application, receive events from the control, and access some of the control's properties.

Tasks illustrated in this walkthrough include:

- Implementing the Windows Forms composite control.
- Implementing the WPF host application.

For a complete code listing of the tasks illustrated in this walkthrough, see Hosting a Windows Forms Composite Control in WPF Sample.

Prerequisites

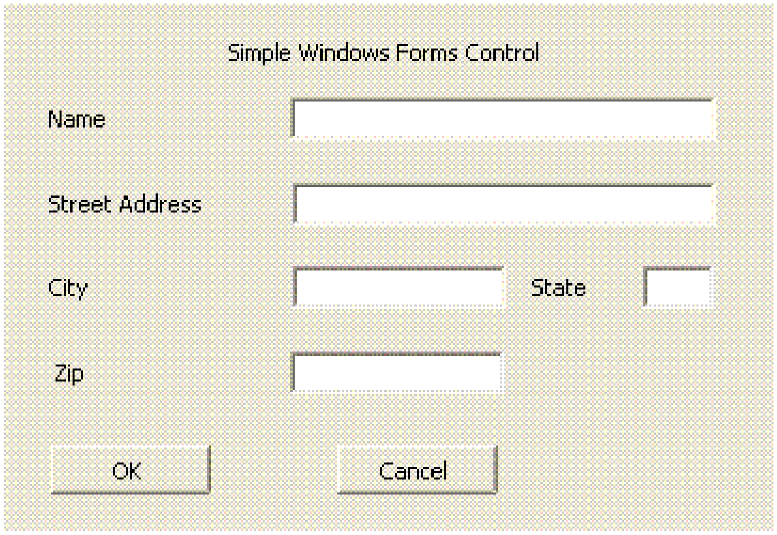
You need the following components to complete this walkthrough:

- Visual Studio 2010.

Implementing the Windows Forms Composite Control

The Windows Forms composite control used in this example is a simple data-entry form. This form takes the user's name and address and then uses a custom event to return that information to the host. The following user's name and address and then uses a custom event to return that information to the host. The following illustration shows the rendered control.

Windows Forms composite control



Creating the Project

To start the project:

1. Launch Microsoft Visual Studio, and open the **New Project** dialog box.
2. In the Window category, select the **Windows Forms Control Library** template.
3. Name the new project **MyControls**.
4. For the location, specify a conveniently named top-level folder, such as **WpfHostingWindowsFormsControl**. Later, you will put the host application in this folder.
5. Click **OK** to create the project. The default project contains a single control named `UserControl1`.
6. In Solution Explorer, rename `UserControl1` to `MyControl1`.

Your project should have references to the following system DLLs. If any of these DLLs are not included by default, add them to the project.

- System
- System.Data
- System.Drawing
- System.Windows.Forms
- System.Xml

Adding Controls to the Form

To add controls to the form:

- Open MyControl1 in the designer.

Add five Label controls and their corresponding TextBox controls, sized and arranged as they are in the preceding illustration, on the form. In the example, the TextBox controls are named:

- txtName
- txtAddress
- txtCity
- txtState
- txtZip

Add two Button controls labeled **OK** and **Cancel**. In the example, the button names are btnOK and btnCancel, respectively.

Implementing the Supporting Code

Open the form in code view. The control returns the collected data to its host by raising the custom OnButtonClick event. The data is contained in the event argument object. The following code shows the event and delegate declaration.

Add the following code to the MyControl1 class.

C#

```
public delegate void MyControlEventHandler(object sender, MyControlEventArgs args
);
public event MyControlEventHandler OnButtonClick;
```

The MyControlEventArgs class contains the information to be returned to the host.

Add the following class to the form.

C#

```
public class MyControlEventArgs : EventArgs
{
    private string _Name;
    private string _StreetAddress;
    private string _City;
    private string _State;
```

```
        private string _Zip;
        private bool _IsOK;

        public MyControlEventArgs(bool result,
                                       string name,
                                       string address,
                                       string city,
                                       string state,
                                       string zip)
        {
            _IsOK = result;
            _Name = name;
            _StreetAddress = address;
            _City = city;
            _State = state;
            _Zip = zip;
        }

        public string MyName
        {
            get { return _Name; }
            set { _Name = value; }
        }
        public string MyStreetAddress
        {
            get { return _StreetAddress; }
            set { _StreetAddress = value; }
        }
        public string MyCity
        {
            get { return _City; }
            set { _City = value; }
        }
        public string MyState
        {
            get { return _State; }
            set { _State = value; }
        }
        public string MyZip
        {
            get { return _Zip; }
            set { _Zip = value; }
        }
        public bool IsOK
        {
            get { return _IsOK; }
            set { _IsOK = value; }
        }
    }
```

When the user clicks the **OK** or **Cancel** button, the Click event handlers create a `MyControlEventArgs` object that contains the data and raises the `OnButtonClick` event. The only difference between the two handlers is the event argument's `IsOK` property. This property enables the host to determine which button was clicked. It is set to **true** for the **OK** button, and **false** for the **Cancel** button. The following code shows the two button handlers.

Add the following code to the MyControl1 class.

C#

```
private void btnOK_Click(object sender, System.EventArgs e)
{

    MyControlEventArgs retvals = new MyControlEventArgs(true,
                                                         txtName.Text,
                                                         txtAddress.Text,
                                                         txtCity.Text,
                                                         txtState.Text,
                                                         txtZip.Text);
    OnButtonClick(this, retvals);
}

private void btnCancel_Click(object sender, System.EventArgs e)
{
    MyControlEventArgs retvals = new MyControlEventArgs(false,
                                                         txtName.Text,
                                                         txtAddress.Text,
                                                         txtCity.Text,
                                                         txtState.Text,
                                                         txtZip.Text);
    OnButtonClick(this, retvals);
}
```

Giving the Assembly a Strong Name and Building the Assembly

For this assembly to be referenced by a WPF application, it must have a strong name. To create a strong name, create a key file with Sn.exe and add it to your project.

1. Open a Visual Studio command prompt. To do so, click the **Start** menu, and then select **All Programs/Microsoft Visual Studio 2010/Visual Studio Tools/Visual Studio Command Prompt**. This launches a console window with customized environment variables.

2. At the command prompt, use the `cd` command to go to your project folder.

3. Generate a key file named MyControls.snk by running the following command.

   ```
   Sn.exe -k MyControls.snk
   ```

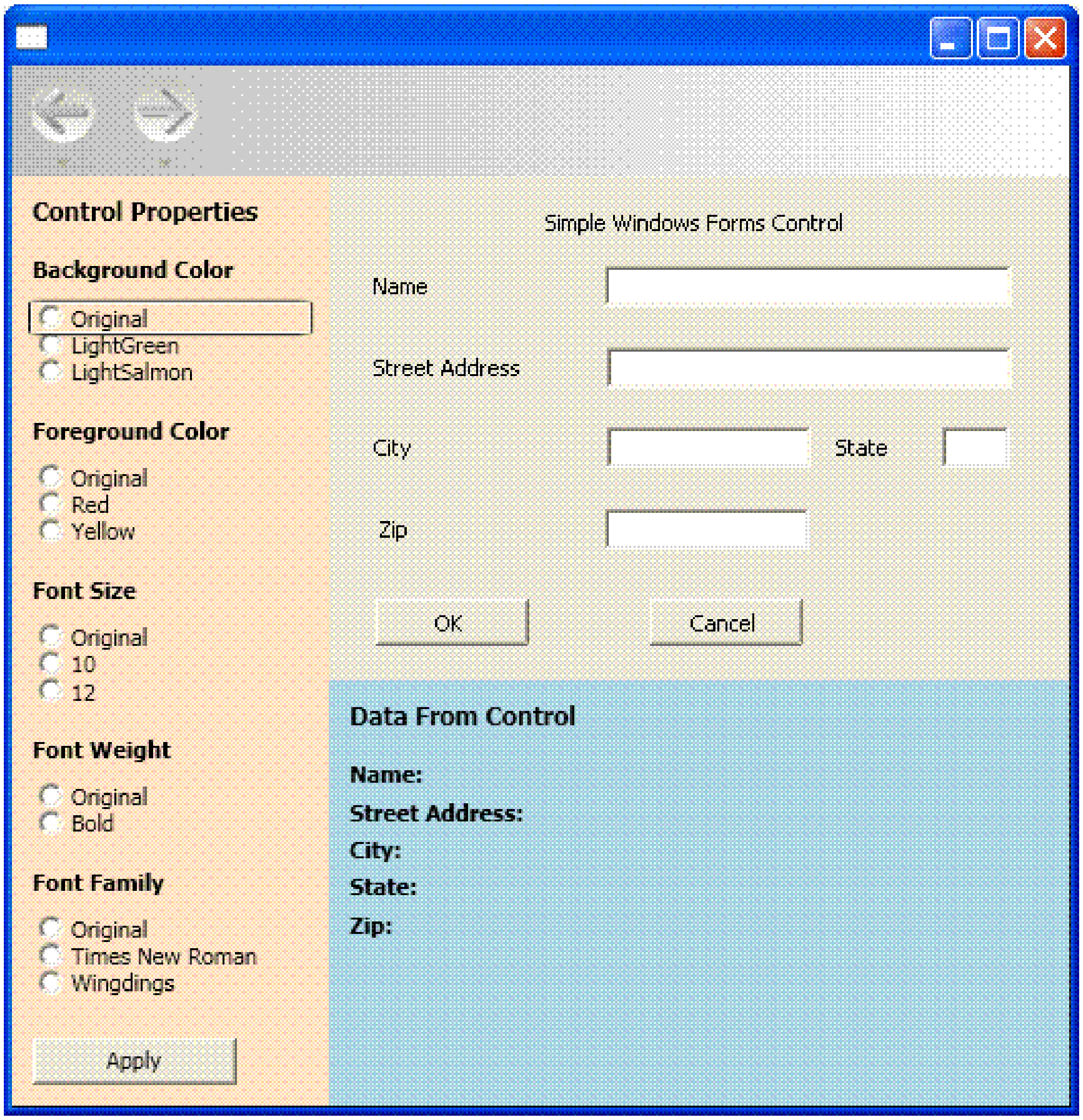
4. To include the key file in your project, right-click the project name in Solution Explorer and then click **Properties**. In the Project Designer, click the **Signing** tab, select the **Sign the assembly** check box and then browse to your key file.

5. Build the solution. The build will produce a DLL named MyControls.dll.

Implementing the WPF Host Application

The WPF host application uses the WindowsFormsHost control to host `MyControl1`. The application handles the `OnButtonClick` event to receive the data from the control. It also has a collection of option buttons that enable you to change some of the control's properties from the WPF application. The following illustration shows the finished application.

The complete application, showing the control embedded in the WPF application



Creating the Project

To start the project:

1. Open Visual Studio, and select **New Project**.

2. In the Window category, select the **WPF Application** template.

3. Name the new project **WpfHost**.

4. For the location, specify the same top-level folder that contains the MyControls project.

5. Click **OK** to create the project.

You also need to add references to the DLL that contains `MyControl1` and other assemblies.

1. Right-click the project name in Solution Explorer and select **Add Reference**.

2. Click the **Browse** tab, and browse to the folder that contains MyControls.dll. For this walkthrough, this folder is MyControls\bin\Debug.

3. Select MyControls.dll, and then click **OK**.

4. Add a reference to the WindowsFormsIntegration assembly, which is named WindowsFormsIntegration.dll.

Implementing the Basic Layout

The user interface (UI) of the host application is implemented in MainWindow.xaml. This file contains Extensible Application Markup Language (XAML) markup that defines the layout, and hosts the Windows Forms control. The application is divided into three regions:

- The **Control Properties** panel, which contains a collection of option buttons that you can use to modify various properties of the hosted control.

- The **Data from Control** panel, which contains several TextBlock elements that display the data returned from the hosted control.

- The hosted control itself.

The basic layout is shown in the following XAML. The markup that is needed to host `MyControl1` is omitted from this example, but will be discussed later.

Replace the XAML in MainWindow.xaml with the following. If you are using Visual Basic, change the class to `x:Class="MainWindow"`.

XAML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="WpfHost.MainWindow"
      xmlns:mcl="clr-namespace:MyControls;assembly=MyControls"
      Loaded="Init">
  <DockPanel>
    <DockPanel.Resources>
```

```
      <Style x:Key="inlineText" TargetType="{x:Type Inline}">
        <Setter Property="FontWeight" Value="Normal"/>
      </Style>
      <Style x:Key="titleText" TargetType="{x:Type TextBlock}">
        <Setter Property="DockPanel.Dock" Value="Top"/>
        <Setter Property="FontWeight" Value="Bold"/>
        <Setter Property="Margin" Value="10,5,10,0"/>
      </Style>
    </DockPanel.Resources>

    <StackPanel Orientation="Vertical"
                DockPanel.Dock="Left"
                Background="Bisque"
                Width="250">

      <TextBlock  Margin="10,10,10,10"
                  FontWeight="Bold"
                  FontSize="12">Control Properties</TextBlock>
      <TextBlock Style="{StaticResource titleText}">Background Color</TextBlock>
      <StackPanel Margin="10,10,10,10">
        <RadioButton Name="rdbtnOriginalBackColor"
                    IsChecked="True"
                    Click="BackColorChanged">Original</RadioButton>
        <RadioButton Name="rdbtnBackGreen"
                    Click="BackColorChanged">LightGreen</RadioButton>
        <RadioButton Name="rdbtnBackSalmon"
                    Click="BackColorChanged">LightSalmon</RadioButton>
      </StackPanel>

      <TextBlock Style="{StaticResource titleText}">Foreground Color</TextBlock>
      <StackPanel Margin="10,10,10,10">
        <RadioButton Name="rdbtnOriginalForeColor"
                    IsChecked="True"
                    Click="ForeColorChanged">Original</RadioButton>
        <RadioButton Name="rdbtnForeRed"
                    Click="ForeColorChanged">Red</RadioButton>
        <RadioButton Name="rdbtnForeYellow"
                    Click="ForeColorChanged">Yellow</RadioButton>
      </StackPanel>

      <TextBlock Style="{StaticResource titleText}">Font Family</TextBlock>
      <StackPanel Margin="10,10,10,10">
        <RadioButton Name="rdbtnOriginalFamily"
                     IsChecked="True"
                    Click="FontChanged">Original</RadioButton>
        <RadioButton Name="rdbtnTimes"
                    Click="FontChanged">Times New Roman</RadioButton>
        <RadioButton Name="rdbtnWingdings"
                    Click="FontChanged">Wingdings</RadioButton>
      </StackPanel>

      <TextBlock Style="{StaticResource titleText}">Font Size</TextBlock>
      <StackPanel Margin="10,10,10,10">
        <RadioButton Name="rdbtnOriginalSize"
                    IsChecked="True"
                    Click="FontSizeChanged">Original</RadioButton>
        <RadioButton Name="rdbtnTen"
                    Click="FontSizeChanged">10</RadioButton>
        <RadioButton Name="rdbtnTwelve"
                    Click="FontSizeChanged">12</RadioButton>
      </StackPanel>

      <TextBlock Style="{StaticResource titleText}">Font Style</TextBlock>
      <StackPanel Margin="10,10,10,10">
        <RadioButton Name="rdbtnNormalStyle"
                     IsChecked="True"
                     Click="StyleChanged">Original</RadioButton>
        <RadioButton Name="rdbtnItalic"
                     Click="StyleChanged">Italic</RadioButton>
      </StackPanel>

      <TextBlock Style="{StaticResource titleText}">Font Weight</TextBlock>
      <StackPanel Margin="10,10,10,10">
        <RadioButton Name="rdbtnOriginalWeight"
                     IsChecked="True"
                   Click="WeightChanged">
          Original
        </RadioButton>
        <RadioButton Name="rdbtnBold"
                   Click="WeightChanged">Bold</RadioButton>
      </StackPanel>
    </StackPanel>

    <WindowsFormsHost Name="wfh"
                     DockPanel.Dock="Top"
                     Height="300">
      <mcl:MyControl1 Name="mc"/>
    </WindowsFormsHost>

    <StackPanel Orientation="Vertical"
                Height="Auto"
                Background="LightBlue">
      <TextBlock Margin="10,10,10,10"
            FontWeight="Bold"
            FontSize="12">Data From Control</TextBlock>
      <TextBlock Style="{StaticResource titleText}">
        Name: <Span Name="txtName" Style="{StaticResource inlineText}"/>
      </TextBlock>
      <TextBlock Style="{StaticResource titleText}">
        Street Address: <Span Name="txtAddress" Style="{StaticResource inlineText
}"/>
      </TextBlock>
      <TextBlock Style="{StaticResource titleText}">
        City: <Span Name="txtCity" Style="{StaticResource inlineText}"/>
      </TextBlock>
      <TextBlock Style="{StaticResource titleText}">
        State: <Span Name="txtState" Style="{StaticResource inlineText}"/>
      </TextBlock>
      <TextBlock Style="{StaticResource titleText}">
        Zip: <Span Name="txtZip" Style="{StaticResource inlineText}"/>
      </TextBlock>
    </StackPanel>
  </DockPanel>
</Window>
```

The first StackPanel element contains several sets of RadioButton controls that enable you to modify various default properties of the hosted control. That is followed by a WindowsFormsHost element, which hosts `MyControl1`. The final StackPanel element contains several TextBlock elements that display the data that is returned by the hosted control. The ordering of the elements and the Dock and Height attribute settings embed the hosted control into the window with no gaps or distortion.

Hosting the Control

The following edited version of the previous XAML focuses on the elements that are needed to host `MyControl1`.

XAML

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      x:Class="WpfHost.MainWindow"
      xmlns:mcl="clr-namespace:MyControls;assembly=MyControls"
      Loaded="Init">


...


<WindowsFormsHost Name="wfh"
                 DockPanel.Dock="Top"
                 Height="300">
  <mcl:MyControl1 Name="mc"/>
</WindowsFormsHost>
```

The `xmlns` namespace mapping attribute creates a reference to the `MyControls` namespace that contains the hosted control. This mapping enables you to represent `MyControl1` in XAML as `<mcl:MyControl1>`.

Two elements in the XAML handle the hosting:

- `WindowsFormsHost` represents the WindowsFormsHost element that enables you to host a Windows Forms control in a WPF application.
- `mcl:MyControl1`, which represents `MyControl1`, is added to the WindowsFormsHost element's child collection. As a result, this Windows Forms control is rendered as part of the WPF window, and you can communicate with the control from the application.

Implementing the Code-Behind File

The code-behind file, MainWindow.xaml.vb or MainWindow.xaml.cs, contains the procedural code that implements the functionality of the UI discussed in the preceding section. The primary tasks are:

- Attaching an event handler to MyControl1's OnButtonClick event.
- Modifying various properties of MyControl1, based on how the collection of option buttons are set.
- Displaying the data collected by the control.

Initializing the Application

The initialization code is contained in an event handler for the window's Loaded event and attaches an event handler to the control's OnButtonClick event.

In MainWindow.xaml.vb or MainWindow.xaml.cs, add the following code to the MainWindow class.

C#

```
private Application app;
private Window myWindow;
FontWeight initFontWeight;
Double initFontSize;
FontStyle initFontStyle;
SolidColorBrush initBackBrush;
SolidColorBrush initForeBrush;
FontFamily initFontFamily;
bool UIIsReady = false;

private void Init(object sender, EventArgs e)
{
    app = System.Windows.Application.Current;
    myWindow = (Window)app.MainWindow;
    myWindow.SizeToContent = SizeToContent.WidthAndHeight;
    wfh.TabIndex = 10;
    initFontSize = wfh.FontSize;
    initFontWeight = wfh.FontWeight;
    initFontFamily = wfh.FontFamily;
    initFontStyle = wfh.FontStyle;
    initBackBrush = (SolidColorBrush)wfh.Background;
    initForeBrush = (SolidColorBrush)wfh.Foreground;
    (wfh.Child as MyControl1).OnButtonClick += new MyControl1.MyControlEventHan
dler(Pane1_OnButtonClick);
    UIIsReady = true;
}
```

Because the XAML discussed previously added MyControl1 to the WindowsFormsHost element's child element collection, you can cast the WindowsFormsHost element's Child to get the reference to MyControl1. You can then use that reference to attach an event handler to OnButtonClick.

In addition to providing a reference to the control itself, WindowsFormsHost exposes a number of the control's properties, which you can manipulate from the application. The initialization code assigns those values to private global variables for later use in the application.

So that you can easily access the types in the MyControls DLL, add the following **Imports** or **using** statement to the top of the file.

C#

```
using MyControls;
```

Handling the OnButtonClick Event

MyControl1 raises the OnButtonClick event when the user clicks either of the control's buttons.

Add the following code to the MainWindow class.

C#

```
//Handle button clicks on the Windows Form control
private void Pane1_OnButtonClick(object sender, MyControlEventArgs args)
{
    txtName.Inlines.Clear();
    txtAddress.Inlines.Clear();
    txtCity.Inlines.Clear();
    txtState.Inlines.Clear();
    txtZip.Inlines.Clear();

    if (args.IsOK)
    {
        txtName.Inlines.Add( " " + args.MyName );
        txtAddress.Inlines.Add( " " + args.MyStreetAddress );
        txtCity.Inlines.Add( " " + args.MyCity );
        txtState.Inlines.Add( " " + args.MyState );
        txtZip.Inlines.Add( " " + args.MyZip );
    }
}
```

The data in the text boxes is packed into the MyControlEventArgs object. If the user clicks the **OK** button, the event handler extracts the data and displays it in the panel below MyControl1.

Modifying the Control's Properties

The WindowsFormsHost element exposes several of the hosted control's default properties. As a result, you can change the appearance of the control to match the style of your application more closely. The sets of option buttons in the left panel enable the user to modify several color and font properties. Each set of buttons has a handler for the Click event, which detects the user's option button selections and changes the corresponding property on the control.

Add the following code to the MainWindow class.

C#

```
private void BackColorChanged(object sender, RoutedEventArgs e)
{
{
    if (sender == rdbtnBackGreen)
        wfh.Background = new SolidColorBrush(Colors.LightGreen);
    else if (sender == rdbtnBackSalmon)
        wfh.Background = new SolidColorBrush(Colors.LightSalmon);
    else if (UIIsReady == true)
        wfh.Background = initBackBrush;
}

private void ForeColorChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnForeRed)
        wfh.Foreground = new SolidColorBrush(Colors.Red);
    else if (sender == rdbtnForeYellow)
        wfh.Foreground = new SolidColorBrush(Colors.Yellow);
    else if (UIIsReady == true)
        wfh.Foreground = initForeBrush;
}

private void FontChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnTimes)
        wfh.FontFamily = new FontFamily("Times New Roman");
    else if (sender == rdbtnWingdings)
        wfh.FontFamily = new FontFamily("Wingdings");
    else if (UIIsReady == true)
        wfh.FontFamily = initFontFamily;
}
private void FontSizeChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnTen)
        wfh.FontSize = 10;
    else if (sender == rdbtnTwelve)
        wfh.FontSize = 12;
    else if (UIIsReady == true)
        wfh.FontSize = initFontSize;
}
private void StyleChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnItalic)
        wfh.FontStyle = FontStyles.Italic;
    else if (UIIsReady == true)
        wfh.FontStyle = initFontStyle;
}
private void WeightChanged(object sender, RoutedEventArgs e)
{
    if (sender == rdbtnBold)
        wfh.FontWeight = FontWeights.Bold;
    else if (UIIsReady == true)
        wfh.FontWeight = initFontWeight;
}
```

Build and run the application. Add some text in the Windows Forms composite control and then click **OK**. The text appears in the labels. Click the different radio buttons to see the effect on the control.

See Also

Tasks

Walkthrough: Hosting a Windows Forms Control in WPF

Reference

ElementHost

WindowsFormsHost

Concepts

Walkthrough: Hosting a WPF Composite Control in Windows Forms

Other Resources

WPF Designer