developmentor

Introduction to WPF

Estimated time for completion: 45 minutes

Goals:

- Learn how to create a simple WPF applications
- Draw shapes
- Use Controls
- Handle events

Overview:

In this lab you will create a simple WPF applications using C# code. When you are finished, your application will look something like:

© 2010 Digital Age Learning. All rights reserved

Part 1 – Creating a WPF application

In this part we will create a basic WPF window and display it on the screen.

Steps:

1. Open Visual Studio 2008 or 2010 and create a new Console Application project. This creates the simplest form of project with a single C# source file and we'll start from there.
2. Since this is a Windows application, change the output type to reflect that.
    a. Right-click on the project and select properties and change the "Output Type" from Console Application to Windows Application.
3. Add references to the core WPF assemblies:
    a. `PresentationCore`
    b. `PresentationFramework`
    c. `WindowsBase`
4. Open the `Program.cs` source file created by the Visual Studio project template - there should be a Main definition present.
5. Add a `STAThread` attribute to the Main entry point.
6. Inside the Main method, create a new `Window` object and set the Title property to whatever you like (the sample will use "Hello, Wpf").
    a. Remember to add a using declaration for `System.Windows` namespace.
7. Set the `Width` property to "480" and `Height` property to "384" - this will size our window to around 4x3 inches.
8. Call the `ShowDialog()` method on your instance of the window and go ahead and run the application.

```
[STAThread]
static void Main(string[] args)
{
   Window w = new Window
   {
      Title = "Hello, Wpf",
      Width = 480,
      Height = 384
   };
   w.ShowDialog();
}
```

© 2010 Digital Age Learning. All rights reserved

Part 2 – Enhancing the appearance of the Window

In this part, we'll make our window a bit prettier plus bring in the `Application` class to manage the window lifetime for us.

Steps:

1. Let's change the project to make it more WPF-ish. First, remove the call to `ShowDialog` - we won't need that anymore.
2. Create a new `Application` object.
3. As the last statement in your Main method, call the `Run` method of the `Application` object and pass it to the `Window` instance. Run the application - it should look and behave exactly the same as before.

```
Application app = new Application();
app.Run(w);
```

4. Create a `System.Windows.Media.LinearGradientBrush` which we will use to fill the background of the window with. There are several constructors, let's use the one which takes two colors and an angle.
    a. Set the first color to `Colors.LightSeaGreen` and the second to `Colors.Yellow`. The `Colors` class holds a series of static definition.
    b. Set the angle to 45.0
    c. Assign the window `Background` property to the new `Brush` and run the application.
    d. **Hint**: Remember to add a using statement for `System.Windows.Media`

```
Window w = new Window
{
   Title = "Hello, Wpf",
   Width = 480,
   Height = 384,
   Background = new LinearGradientBrush(Colors.LightSeaGreen,
                                        Colors.Yellow, 45.0)
};
```

5. Now let's add some content to the window! We could do this right inside our Main method, but let's use an event handler instead.
    a. Tie a method to the `Loaded` event of the Window instance. This event is fired when the window is being created and is raised just prior to it being rendered for the first time.
    b. The event signature is similar to the one you are probably already used to:

```
void Method(object sender, RoutedEventArgs e);
```

© 2010 Digital Age Learning. All rights reserved

Image

```
w.Loaded += w_Loaded;
```

6. In your event handler, create an `Ellipse` object. You will need the `System.Windows.Shapes` namespace for this.
    a. Set the `Stroke` to `Brushes.Red` and the `StrokeThickness` to 1 to get a border around the shape.

```
static void w_Loaded(object sender, RoutedEventArgs e)
{
   Ellipse ellipse = new Ellipse
    { Stroke = Brushes.Red, StrokeThickness = 1 };
}
```

7. Let's fill the ellipse with an image. Recall that fills and backgrounds require a Brush object, so we'll use an `ImageBrush` to satisfy that requirement.
    a. Create an `ImageBrush` object - note that the non-default constructor takes an `ImageSource` object.
    b. `ImageSource` objects cannot be created directly since the constructor is private, however the `ImageSourceConverter` will load images from files and return them to us.
    c. Create an `ImageSourceConverter` and call the `ConvertFrom` method passing it a full path and filename to an image. Any image will do, the sample code here will select one from the `C:\Windows\Web\WallPaper` directory. Remember that you will need to escape the backslashes or prefix the string with a "`@`" in C#.

Note that the above path is specific to Windows Vista – older versions of Windows may have the images in different places. Any bitmap image will do, so feel free to grab one from "My Pictures" or even download one from the Internet if you like.

    d. Cast the result from the `ImageSourceConverter` into an `ImageSource` and supply that to either the constructor of the `ImageBrush` or, if you are using the default constructor, set the `ImageSource` property of the brush.
    e. Finally, assign the `Fill` property of the Ellipse to the created brush.

```
ImageBrush brush = new ImageBrush();
ImageSourceConverter isc = new ImageSourceConverter();
brush.ImageSource = (ImageSource) isc.ConvertFrom(
                                  @"c:\windows\web\wallpaper\img1.jpg");
ellipse.Fill = brush;
```

8. The last step is to make the Ellipse part of the scene. In order to do this, we need access to the `Window` object we created in the Main method. There are three ways to get at this:
    a. Store the window as a field of the class.

© 2010 Digital Age Learning. All rights reserved

   b. Cast the `sender` parameter to the Window (since it's the one firing the event).
   c. Get it from the `Application.Current` object. It's stored as the `MainWindow` property because it was the first created window in our application.
9. Retrieve the window object through any of the three available mechanisms and assign the ellipse to the `Content` property of the window.
10. Run the application - notice how the ellipse fills the window even though we didn't specify a Width or Height for the shape itself. Resize the window and notice that it is automatically sizing the shape too. This is the default behavior for content added to a Window - to take up all available space. WPF manages all of that and redraws the shape and background as necessary.

Part 3 – Creating event handlers

In this part, we'll add some event handler logic to the application to react to user events.

Note: The following steps require a minimum of **.NET 3.5 SP1** – this is where the hardware pixel shader support (Effect) was added to WPF. If you are on an earlier version of .NET, you can use the **BitmapEffect** property and **OuterGlowBitmapEffect** class. This style was deprecated (and actually does not work in .NET 4.0+) because it renders in software and is fairly slow.

Steps:

1. In the `Window.Loaded` handler, add two new event handlers to the ellipse object itself - a `MouseEnter` and a `MouseLeave` handler.

```
ellipse.MouseEnter +=
       new System.Windows.Input.MouseEventHandler(ellipse_MouseEnter);
ellipse.MouseLeave +=
       new System.Windows.Input.MouseEventHandler(ellipse_MouseLeave);
```

2. In the `MouseEnter` handler, cast the sender to an `Ellipse` to get access to the shape.
3. Next, create a `DropShadowEffect` object from the `System.Windows.Media.Effects` namespace.
   a. Set the `Color` to `Colors.DeepPink` and the `BlurRadius` to "30" and the `ShadowDepth` to "0". This will create an "outer glow" effect on the shape.
   b. Assign the pixel shader effect object to the Ellipse's `Effect` property.

```
static void ellipse_MouseEnter(object sender,
                               System.Windows.Input.MouseEventArgs e)
{
    Ellipse ellipse = (Ellipse)sender;
    ellipse.Effect = new DropShadowEffect {
        Color = Colors.DeepPink, BlurRadius = 30, ShadowDepth = 0 };
```

© 2010 Digital Age Learning. All rights reserved

```
}
```

4. In the `MouseLeave` handler, reset the Ellipse's `Effect` property to `null`.

```
static void ellipse_MouseLeave(object sender,
                               System.Windows.Input.MouseEventArgs e)
{
    Ellipse ellipse = (Ellipse)sender;
    ellipse.Effect = null;
}
```

5. Run the application - notice now that when you mouse over the shape, it gets a "glow" effect without any need for you to redraw or invalidate the shape.

© 2010 Digital Age Learning. All rights reserved

Part 4 – Using Controls

In this section, we'll move the Ellipse into a ListBox along with a Button which will play a sound when it is clicked – showing you that content in WPF is a far more general concept than what you may be used to.

Steps:

1. In the `Loaded` event just after you are creating the `Ellipse` shape, create a `ListBox` element – you will need the `System.Windows.Controls` namespace.
    a. Add the `Ellipse` to the `ListBox.Items` collection (call `Add` on the `Items` property of the `ListBox`).
    b. Change the `Window.Content` property to point to the `ListBox` instead of the `Ellipse`.
2. Add a `Button` element after the `ListBox`.
    a. Set the `Width` to "200"
    b. Set the `Height` to "50"
    c. Set the `Content` to "I Dare you to Click Me"
    d. Add the `Button` to the `ListBox Items` collection.

```
ListBox listBox = new ListBox();
listBox.Items.Add(ellipse);

Button button = new Button();
button.Width = 200;
button.Height = 50;
button.Content = "I Dare You to Click Me";

listBox.Items.Add(button);

Window win = (Window)sender;
win.Content = listBox;
```

3. Run the application and observe the results.
    a. Notice that only the button shows up? This is because the `ListBox`, unlike the `Window`, is **not automatically sizing** the `Ellipse` to the available space. To fix this, set the `Width` and `Height` of the `Ellipse` to "200". Run the application again to verify you can see both the ellipse *and* the button.

```
ellipse.Width = 200;
ellipse.Height = 200;
```

    b. Notice as well that we've lost our gradient background? That's because the `ListBox` is now painting over it – you can either move the brush to the

© 2010 Digital Age Learning. All rights reserved

`ListBox` element, or set the `ListBox.Background` to null to stop it from painting a background.

```
listBox.Background = null;
```

4. Notice that we can add different types of items to the `ListBox`? This is one of the great features of WPF – content isn't just about text, it's about anything you can display. It's still a `ListBox` – try selecting the item to verify this.
5. As a final step, add a Click handler to the button – point it at a method OnClick.

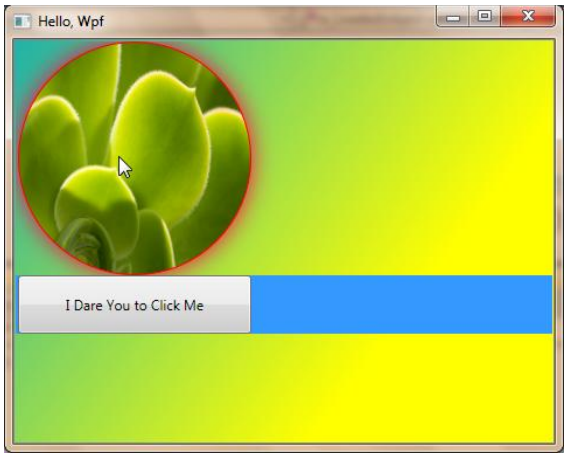
```
button.Click += OnClick;
```

6. In the event handler implementation, play a sound using the `System.Media.SoundPlayer` class.
   a. There are a bunch of .WAV files contained in the `c:\windows\media` directory - any of these will suffice, we will use "tada.wav" in the code sample.

```
static void OnClick(object sender, RoutedEventArgs e)
{
    SoundPlayer sp = new SoundPlayer(@"c:\windows\media\tada.wav");
    sp.Play();
}
```

If the machine does not have a sound card or speakers, output a message box to the screen instead using the `MessageBox.Show` method.

7. Run the application and verify it plays the sound file or shows the message box when you click the button. The final application should look something like:



© 2010 Digital Age Learning. All rights reserved

Solution

The full solution for this lab is available in the **after** folder in the lab directory. It has versions for both VS2008 SP1 and VS2010 that include local resources (an image and sound file).

© 2010 Digital Age Learning. All rights reserved