How to write a templated control

An excellent source for Silverlight 2 templated control authoring can be found on Shawn Burke's blog.

Define the template parts

Identify the parts of your control that you will need to interact with programmatically.

```
[TemplatePart(Name = PathButton.ElementRootName, Type = typeof(FrameworkElement))]
[TemplatePart(Name = PathButton.ElementBackgroundName, Type = typeof(Rectangle))]
[TemplatePart(Name = PathButton.ElementPathName, Type = typeof(Path))]
[TemplatePart(Name = PathButton.ElementPathHoverName, Type = typeof(Path))]
public class PathButton : ButtonBase
{
        internal const string ElementRootName = "RootElement";
        internal const string ElementBackgroundName = "BackgroundElement";
        internal const string ElementPathName = "PathElement";
        internal const string ElementPathHoverName = "PathHoverElement";
```

Following the example of the Silverlight 2 control library, we also use string constants for element names. We do this since we will be referencing them again later. We also define the type of element each part is meant to represent.

Define the visual states

Identify the transition animations your control will be using in the form of visual states defined within the VisualStateManager.

```
[TemplateVisualState(Name = "Normal", GroupName = "CommonStates")]
[TemplateVisualState(Name = "MouseOver", GroupName = "CommonStates")]
public class PathButton : ButtonBase
{
```

Set the DefaultStyleKey

The DefaultStyleKey property defines the default style for your control. This is usually set within the constructor of your control.

```
public PathButton()
{
        DefaultStyleKey = typeof(PathButton);
}
```

Write the OnApplyTemplate logic

Once a control is loaded, the UI is generated from the template defined by the DefaultStyleKey property in the OnApplyTemplate method. You should override this method in order to obtain references to the element parts that you have declared above.

Whether you use member variables or wrap them in scoped properties is your choice.

```
public override void OnApplyTemplate()
{
        base.OnApplyTemplate();

        object root = GetTemplateChild(ElementRootName);
        Debug.Assert(typeof(FrameworkElement).IsInstanceOfType(root) || (root == null),
                "The template part RootElement is not an instance of FrameworkElement!");
        RootElement = root as FrameworkElement;
```

```
        object background = GetTemplateChild(ElementBackgroundName);
        Debug.Assert(typeof(Rectangle).IsInstanceOfType(background) || (background == null),
                "The template part BackgroundElement is not an instance of Rectangle!");
        BackgroundElement = background as Rectangle;

        object path = GetTemplateChild(ElementPathName);
        Debug.Assert(typeof(Path).IsInstanceOfType(path) || (path == null),
                "The template part PathElement is not an instance of Path!");
        PathElement = path as Path;

        object pathHover = GetTemplateChild(ElementPathHoverName);
        Debug.Assert(typeof(Path).IsInstanceOfType(pathHover) || (pathHover == null),
                "The template part PathHoverElement is not an instance of Path!");
        PathHoverElement = pathHover as Path;
}
```

In the example above, we used scoped properties to wrap the access to template parts. We also added additional code for debugging purposes. The key to this section is in calling the GetTemplateChild method, which returns an object that can be cast to the appropriate type.

Create your dependency properties

Identify the properties you want to use for template binding. This allows you to pass through settings from a control to the UI within the template.

```
public double PathWidth
{
        get { return (double)GetValue(PathWidthProperty); }
        set { SetValue(PathWidthProperty, value); }
}

public static readonly DependencyProperty PathWidthProperty =
        DependencyProperty.Register(
                "PathWidth",
                typeof(double),
                typeof(PathButton),
                null);
```

Creating your default template

You create the default template for your control within a XAML file called generic.xaml (which resides in the Themes folder).

```
<Style TargetType="local:PathButton">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="local:PathButton">
        <!-- template xaml -->
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Managing state changes

In Silverlight 2 we use the VisualStateManager to handle visual state changes. You specify the VisualStateManager XAML within the root element of your template.

```
<vsm:VisualStateManager.VisualStateGroups>
        <vsm:VisualStateGroup x:Name="CommonStates">
                <vsm:VisualStateGroup.Transitions>
                        <vsm:VisualTransition
                                To="MouseOver"
                                Duration="0:0:0.2" />
                        <vsm:VisualTransition
```

```
                        From="MouseOver"
                        To="Normal"
                        Duration="0:0:0.2">
                        <Storyboard>
                                <DoubleAnimation
                                        Storyboard.TargetName="PathElement"
                                        Storyboard.TargetProperty="Opacity"
                                        To="1" />
                                <DoubleAnimation
                                        Storyboard.TargetName="PathHoverElement"
                                        Storyboard.TargetProperty="Opacity"
                                        To="0" />
                        </Storyboard>
                </vsm:VisualTransition>
        </vsm:VisualStateGroup.Transitions>
        <vsm:VisualState x:Name="Normal" />
        <vsm:VisualState x:Name="MouseOver">
                <Storyboard>
                        <DoubleAnimation
                                Storyboard.TargetName="PathElement"
                                Storyboard.TargetProperty="Opacity"
                                To="0" />
                        <DoubleAnimation
                                Storyboard.TargetName="PathHoverElement"
                                Storyboard.TargetProperty="Opacity"
                                To="1" />
                </Storyboard>
        </vsm:VisualState>
    </vsm:VisualStateGroup>
</vsm:VisualStateManager.VisualStateGroups>
```

You may trigger these state changes by calling the GoToState method, specifying the appropriate state and whether or not to use defined transitions.

```
VisualStateManager.GoToState(this, "MouseOver", true);
```