developmentor

Styles

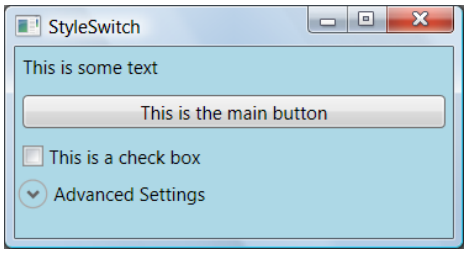
Estimated time for completion: 60 minutes

Goals:

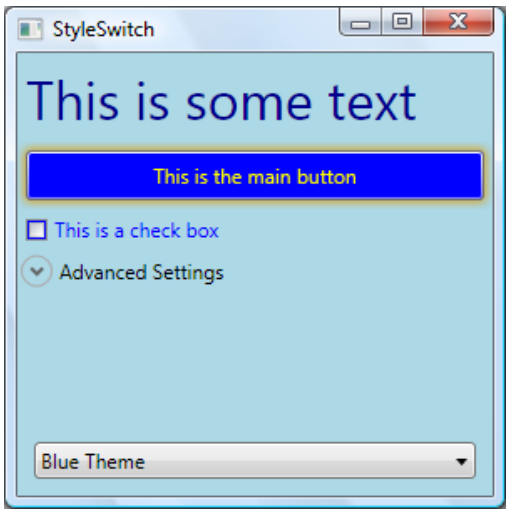
- Utilize styles to define the visual appearance of controls in a single spot
- Set properties using styles
- Apply styles to a set of controls
- Switch styles dynamically by defining multiple style resources.

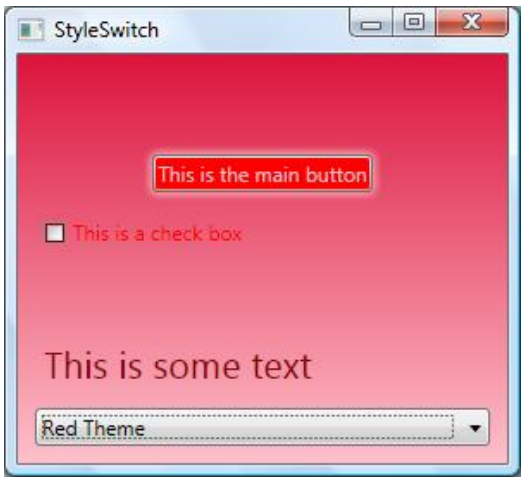
Overview:

In this lab, you'll start with a simple pre-built application and will use styles to change the look and feel. You will start with an application that looks like:



Then through some applied styles, you will convert it to have different "themes" – a blue and red theme:





Part 1 – Creating the initial styles

In this part, you'll define a basic style for the application. The style will change the color scheme and the overall visual structure of the window.

Steps:

1. Open the **before** solution **StyleSwitch.sln** and navigate to the main window (`Window1.xaml`).
2. In the Resources of the `StackPanel`, add a new `Style` element.
    a. Set the key to "BasicSettings".
    b. Add a setter for the `Control.Margin` property and set the value to "5".
    c. Add a setter for the `Control.Padding` property and set the value to "5".
    d. Add a setter for the `Control.Background` property and set the value to "Blue".
    e. Add a setter for the `Control.Foreground` property and set the value to "White".

```
<StackPanel.Resources>
  <Style x:Key="BasicSettings">
    <Setter Property="Control.Margin" Value="5" />
    <Setter Property="Control.Padding" Value="5" />
    <Setter Property="Control.Background" Value="Blue" />
    <Setter Property="Control.Foreground" Value="White" />
  </Style>
</StackPanel.Resources>
```

3. Add another `Style` element.
    a. Set the `TargetType` to "{x:Type Button}".
    b. Set the `BasedOn` property to "{StaticResource BasicSettings}".
4. Add another `Style` element.
    a. Set the `TargetType` to "{x:Type CheckBox}".
    b. Set the `BasedOn` property to "{StaticResource BasicSettings}".
    c. Set the `Foreground` property to "Blue".
5. Add another `Style` element.
    a. Set the `TargetType` to "{x:Type ToggleButton}".
    b. Set the `BasedOn` property to "{StaticResource BasicSettings}".
6. Next, add a named `Style` called "TitleText".
    a. Set the `BasedOn` property to "{StaticResource BasicSettings}".

b. Since this style does not specify a target type, we must fully qualify each property setter – so, first set the `TextBlock.Foreground` property to "DarkBlue"

c. Set the `TextBlock.FontSize` to "24pt".

7. Modify the `TextBlock` (1st element in the `StackPanel`) to use this new style

   a. Set the `Style` property to "{DynamicResource TitleText}".

8. The markup should look like:

```
<StackPanel>
   <StackPanel.Resources>
     <Style x:Key="BasicSettings">
       <Setter Property="Control.Margin" Value="5" />
       <Setter Property="Control.Padding" Value="5" />
       <Setter Property="Control.Background" Value="Blue" />
       <Setter Property="Control.Foreground" Value="White" />
     </Style>

     <Style TargetType="{x:Type Button}"
            BasedOn="{StaticResource BasicSettings}">
     </Style>

     <Style TargetType="{x:Type CheckBox}"
            BasedOn="{StaticResource BasicSettings}">
        <Setter Property="Foreground" Value="Blue" />
     </Style>

     <Style TargetType="{x:Type ToggleButton}"
            BasedOn="{StaticResource BasicSettings}">
     </Style>

     <Style x:Key="TitleText" BasedOn="{StaticResource BasicSettings}">
        <Setter Property="TextBlock.Foreground" Value="DarkBlue" />
        <Setter Property="TextBlock.FontSize" Value="24pt" />
     </Style>
   </StackPanel.Resources>

   <TextBlock Style="{DynamicResource TitleText}">
           This is some text</TextBlock>
   <Button>This is the main button</Button>
   <CheckBox>This is a check box</CheckBox>
   <Expander Header="Advanced Settings">
      <ToggleButton>Turn on Advanced settings!</ToggleButton>
   </Expander>
</StackPanel>
```

9. Run the application – it should now have the buttons in blue and a larger title text.

10. Next, add a `Style` resource to the `StackPanel` and apply it specifically to the main button (the button with the text "This is the main button").
    a. Base it on the "BasicSettings" style.
    b. Set the `Foreground` to "Yellow"
    c. Add a `DropShadowEffect` with a `Color` of "GoldenRod", a `BlurRadius` of "10" and a `ShadowDepth` of "0". This is applied to the `Effect` property of the MainButton:

```
<Style TargetType="{x:Type Button}" x:Key="MainButton"
       BasedOn="{StaticResource BasicSettings}">
    <Setter Property="Foreground" Value="Yellow" />
    <Setter Property="Effect">
       <Setter.Value>
          <DropShadowEffect Color="Goldenrod" BlurRadius="10"
                            ShadowDepth="0" />
       </Setter.Value>
    </Setter>
</Style>
...
<Button Style="{DynamicResource MainButton}">This is the Main Button</Button>
```

11. Run the application again and note that now that one button has the new style, but the advanced button (in the expander) does not.

Part 2 – Styling applications with separate XAML files

In this part, you'll move the style definitions to a separate XAML file.

Steps:

1. Currently the style definitions are part of the `StackPanel`. You could also move them to the `Window`, or even to the `Application` level and WPF would continue to find them. However, our goal with this part is to move them to a separate XAML file altogether and get WPF to find them.
2. Add a new Resource Dictionary to the project (Right click on the project, Add->New, select "Resource Dictionary").
    a. Name it "BlueTheme.xaml".
3. Move the style definitions from the `StackPanel` into this new .XAML file. It should already have a `ResourceDictionary` key you can place all the styles into.

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <Style x:Key="BasicSettings">
```

```
    <Setter Property="Control.Margin" Value="5" />
    <Setter Property="Control.Padding" Value="5" />
    <Setter Property="Control.Background" Value="Blue" />
    <Setter Property="Control.Foreground" Value="White" />
  </Style>
  ...
</ResourceDictionary>
```

4. Open the app.xaml file – it should have an `Application.Resources` tag in it.
5. Add a `ResourceDictionary` tag into the `Application.Resources`.
    a. Set the `Source` property to "BlueTheme.xaml".

```
<Application x:Class="StyleSwitch.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Window1.xaml">

    <Application.Resources>
      <ResourceDictionary Source="BlueTheme.xaml" />
    </Application.Resources>

</Application>
```

6. Run the application – note that it continues to use the new style – even though the style is now defined in a separate file. It is now being included into the resources through the `Application.Resources` and WPF walks the resource tree looking for the specified styles.
7. If you have more than one resource dictionary (as shown in the slides if you want to refer back to them), you would then need to add them into a merged dictionary collection and add THAT into the application resources – for example:
    a. **Note: this is an example only – do not modify your code here.**

```
<Application x:Class="StyleSwitch.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Window1.xaml">

    <Application.Resources>
      <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
          <ResourceDictionary Source="BlueTheme.xaml" />
          <ResourceDictionary Source="BlueTheme.xaml" />
        </ResourceDictionary.MergedDictionaries>
       </ResourceDictionary>
    </Application.Resources>
```

```
</Application>
```

Part 3 – Dynamically changing the style

In this part, you'll create a second style for the application and change it dynamically.

Steps:

1. **Remove** the `ResourceDictionary` you added above from the app.xaml.
2. Add the following function into the app.xaml.cs file – it will dynamically load a style included in the application:

```
public void SkinTheApp(string skinName)
{
   // Load the skin
   ResourceDictionary rd = new ResourceDictionary();
   rd.Source = new Uri(skinName, UriKind.Relative);

   // Remove the old skin
   if (Resources.MergedDictionaries.Count > 0)
      Resources.MergedDictionaries.Remove(
                  Resources.MergedDictionaries[0]);
   // Add the new skin
   Resources.MergedDictionaries.Add(rd);
}
```

3. Next, add an Application constructor into the app.xaml.cs file and call the `SkinTheApp` method with "bluetheme.xaml" as the parameter:

```
public App()
{
   SkinTheApp("bluetheme.xaml");
}
```

4. Run the application and make sure your theme continues to work.
5. Remove the constructor call – we're going to move it into the window now.
6. Add a new Resource Dictionary to the project (see the previous part for instructions on this).
    a. Call it "RedTheme.xaml".
    b. Copy the BlueTheme.xaml styles into the new theme file.
    c. Change all occurrences of "Blue" as a color to "Red".
    d. For the MainButton style, set the glow color (on the drop shadow) to "White" and `Foreground` to "Pink".

e. For the TitleText style, set the `FontSize` to "16pt" to make it smaller.

7. Open the window1.xaml file and add row definitions for the Grid so that it has two rows.
   a. The first row should have a height of "*".
   b. The second row should have a height of "Auto".

```
<Grid.RowDefinitions>
   <RowDefinition />
   <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

8. Add a `ComboBox` as the last item in the `Grid` and set the `Grid.Row` to be "1".
   a. Set the `Margin` property to "10".
   b. Add an event handler for the `SelectionChanged` event – name it "OnThemeChanged". We'll create the function in a moment.
   c. Add a `ComboBoxItem` with a `Content` of "Blue Theme" and a `Tag` of "bluetheme.xaml".
   d. Add a `ComboBoxItem` with a `Content` of "Red Theme" and a `Tag` of "redtheme.xaml".
   e. Finally, set the `SelectedIndex` to "0" on the `ComboBox`.

```
<ComboBox Margin="10" Grid.Row="1" SelectedIndex="0"
          SelectionChanged="OnThemeChanged">
   <ComboBoxItem Content="Blue Theme" Tag="bluetheme.xaml" />
   <ComboBoxItem Content="Red Theme" Tag="redtheme.xaml" />
</ComboBox>
```

9. Open the window1.xaml.cs file and add a handler for "OnThemeChanged".
   a. In the handler, cast the sender to a `ComboBox`.
   b. Using the combo box, retrieve the `SelectedItem` property and cast it to a `ComboBoxItem`.
   c. Cast the `Tag` property on the `ComboBoxItem` to a string and call the `SkinTheApp` method you added to the application class.
      i. **Hint:** you can retrieve the `Application` instance using `Application.Current`, you will need to cast it to the appropriate type. If you need more help, check the code box below step 10.
10. Run the application – try changing the combo box selection and see how the theme is changed dynamically.

```
void OnThemeChanged(object sender, RoutedEventArgs e)
{
   ComboBox cb = (ComboBox)sender;
```

```
    string skinName = (string)((ComboBoxItem)cb.SelectedItem).Tag;
    ((App)Application.Current).SkinTheApp(skinName);
}
```

Part 4 – Going beyond colors

In this part, you'll add a couple more items into the style to affect the layout of the window.

Steps:

1. Open the window1.xaml file and locate the expander control.
2. Set a named style onto the `Expander` called "AdvancedSettings". Use a `DynamicResource` tag to set the style.

```
<Expander Style="{DynamicResource AdvancedSettings}"
          Header="Advanced Settings">
   <ToggleButton>Turn on Advanced settings!</ToggleButton>
</Expander>
```

3. Add a `Background` property on the window and load it from a resource named "AppBackground".

```
<Window x:Class="StyleSwitch.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="StyleSwitch" Height="300" Width="300"
    Background="{DynamicResource AppBackground}">
```

4. Open the bluetheme.xaml file and add a `SolidColorBrush` with a key of "AppBackground". Set the color of the brush to be "LightBlue".

```
<SolidColorBrush x:Key="AppBackground" Color="LightBlue" />
```

5. Open the redtheme.xaml file and add a new style called "AdvancedSettings".
    a. Add a setter which changes the `Control.Visibility` property to "Collapsed".

```
<Style x:Key="AdvancedSettings">
   <Setter Property="Control.Visibility" Value="Collapsed" />
</Style>
```

6. Locate the main button style.
    a. Add a setter which sets the `HorizontalAlignment` to "Center".
7. Locate the TitleText style and add a `TextBlock.RenderTransform` setter
    a. Set the value to a `TranslateTransform` with a Y value of "150".

```
<Style x:Key="TitleText" BasedOn="{StaticResource BasicSettings}">
    <Setter Property="TextBlock.Foreground" Value="DarkRed" />
    <Setter Property="TextBlock.FontSize" Value="16pt" />
```

```
    <Setter Property="TextBlock.RenderTransform">
      <Setter.Value>
        <TranslateTransform Y="150" />
      </Setter.Value>
    </Setter>
  </Style>
```

8. Add a `LinearGradientBrush` with a key of "AppBackground". Have the colors go from "Crimson" to "Pink". You can choose any direction you like – the lab has it vertically.

```
<LinearGradientBrush x:Key="AppBackground" StartPoint="0,0" EndPoint="0,1">
    <GradientStop Offset="0" Color="Crimson" />
    <GradientStop Offset="1" Color="Pink" />
</LinearGradientBrush>
```

9. Run the application and switch themes – note now how the advanced section actually disappears because we've changed the visibility. This technique could be used to remove sections of the window.

Solution

There is a full solution to this lab in the after folder – note that there are separate solutions for Visual Studio 2008 SP1 and Visual Studio 2010.