

STYLES AND BEHAVIORS

Chapter 11 of Pro WPF : By Matthew MacDonald

Assist Lect. Wadhah R. Baiee . College
of IT – Univ. of Babylon - 2014

Introduction

□ **Styles**

- They are an essential tool for organizing and reusing for formatting choices.
- Rather than filling your XAML with repetitive markup to set details such as margins, padding, colors, and fonts, you can create a set of styles that encompass all these details.
- You can then apply the styles where you need them by setting a single property.

Introduction

□ Behaviors

- They are a more ambitious tool for reusing user interface code.
- The basic idea is that a behavior encapsulates a common bit of UI functionality (for example, the code that makes an element draggable).
- If you have the right behavior, you can attach it to any element with a line or two of XAML markup, saving you the effort of writing and debugging the code yourself.

Style Basics

- Like CSS (Cascading Style Sheets) , WPF styles allow you to define a common set of formatting characteristics and apply them throughout your application to **ensure consistency**.
- You may want to give all buttons a consistent typeface and text size independent from the font settings that are used in other elements.
- In this case, you need a way to define these details in one place and reuse them wherever they apply.

Style Basics

- Styles provide the perfect solution. You can define a single style that wraps all the properties you want to set. Here's how:

```
<Window.Resources>
  <Style x:Key="BigFontButtonStyle">
    <Setter Property="Control.FontFamily" Value="Times New Roman" />
    <Setter Property="Control.FontSize" Value="18" />
    <Setter Property="Control.FontWeight" Value="Bold" />
  </Style>
</Window.Resources>
```

Style Basics

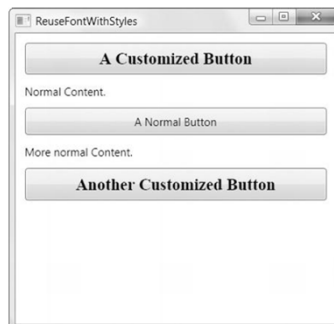
- The style plugs into an element through the element's Style property (which is defined in the base FrameworkElement class).
- For example, to configure a button to use the style you created previously, you'd point the button to the style resource like this:

```
<Button Padding="5" Margin="5" Name="cmd"
  Style="{StaticResource BigFontButtonStyle}">
  A Customized Button
</Button>
```

Style Basics

- Of course, you could also set a style programmatically. All you need to do is pull the style out of the closest Resources collection using the familiar FindResource() method.

```
cmd.Style = (Style)cmd.FindResource("BigFontButtonStyle");
```



Style Basics

- In the previous example, the font settings were organized into a style named BigFontButtonStyle.
- If you decide later that your big-font buttons also need more padding and margin space, you can add setters for the Padding and Margin properties as well.
- The Setters collection is the most important property of the Style class. But there are five key properties altogether.

Style Basics

Table 11-1. Properties of the Style Class

- Setters
- Triggers
- Resources
- BasedOn
- TargetType

Setting Properties

- In some cases, you won't be able to set the property value using a simple attribute string. For example, an ImageBrush object can't be created with a simple string. !!!!!

```
<Style x:Key="HappyTiledElementStyle">
  <Setter Property="Control.Background">
    <Setter.Value>
      <ImageBrush TileMode="Tile"
        ViewportUnits="Absolute" Viewport="0 0 32 32"
        ImageSource="happyface.jpg" Opacity="0.3">
      </ImageBrush>
    </Setter.Value>
  </Setter>
</Style>
```

- If you want to reuse the same image brush in more than one style , you can define it as a resource and then use that resource in your style.

Setting Properties

- consider the following version of the BigFontButton style, which replaces the references to the Control class with references to the Button class:

```
<Style x:Key="BigFontButtonStyle">
  <Setter Property="Button.FontFamily" Value="Times New Roman" />
  <Setter Property="Button.FontSize" Value="18" />
  <Setter Property="Button.FontWeight" Value="Bold" />
</Style>
```

Setting Properties

- If you **substitute this style** in the same example you'll get the **same result**.
- **So, why the difference?**
- In this case, the distinction is how WPF handles other classes that **may include the same FontFamily, FontSize, and FontWeight properties but that don't derive from Button**.
- For example, if you apply this version of the **BigFontButton style to a Label control**, it has no effect.
- **WPF simply ignores the three properties** because they don't apply.
- But if you use **the original style, the font properties will affect the label because the Label class** derives from Control.

Setting Properties

- If all your properties are intended for the same element type, you can set the `TargetType` property of the `Style` object to indicate the class to which your properties apply.

```
<Style x:Key="BigFontButtonStyle" TargetType="Button">
  <Setter Property="FontFamily" Value="Times New Roman" />
  <Setter Property="FontSize" Value="18" />
  <Setter Property="FontWeight" Value="Bold" />
</Style>
```

Attaching Event Handlers

- You can also create a collection of `EventSetter` objects that wire up events to specific event handlers. Here's an example that attaches the event handlers for the `MouseEnter` and `MouseLeave` events:

```
<Style x:Key="MouseOverHighlightStyle">
  <EventSetter Event="TextBlock.MouseEnter" Handler="element_MouseEnter" />
  <EventSetter Event="TextBlock.MouseLeave" Handler="element_MouseLeave" />
  <Setter Property="TextBlock.Padding" Value="5"/>
</Style>
```

Here's the event handling code:

```
private void element_MouseEnter(object sender, MouseEventArgs e)
{
    ((TextBlock)sender).Background =
        new SolidColorBrush(Colors.LightGoldenrodYellow);
}

private void element_MouseLeave(object sender, MouseEventArgs e)
{
    ((TextBlock)sender).Background = null;
}
```

Attaching Event Handlers

- If you want to apply a mouseover effect to a large number of elements (for example, you want to change the background color of an element when the mouse moves overtop of it), you need to add the `MouseEnter` and `MouseLeave` event handlers to each element.

```
<TextBlock Style="{StaticResource MouseOverHighlightStyle}">
  Hover over me.
</TextBlock>
```

- Event setters aren't a good choice when handling an event that uses bubbling.
- In this situation, it's usually easier to handle the event you want on a higher-level element.

The Many Layers of Styles

- Imagine you want to give a group of controls the same font without applying the same style to each one.
- In this case, you may be able to place them in a single panel (or another type of container) and set the style of the container.
- As long as you're setting properties that use the property value inheritance feature, these values will flow down to the children. Properties that use this model include `IsEnabled`, `IsVisible`, `Foreground`, and all the font properties.
- You can use this sort of style inheritance by setting the `BasedOn` attribute of a style. For example, consider these two styles:

The Many Layers of Styles

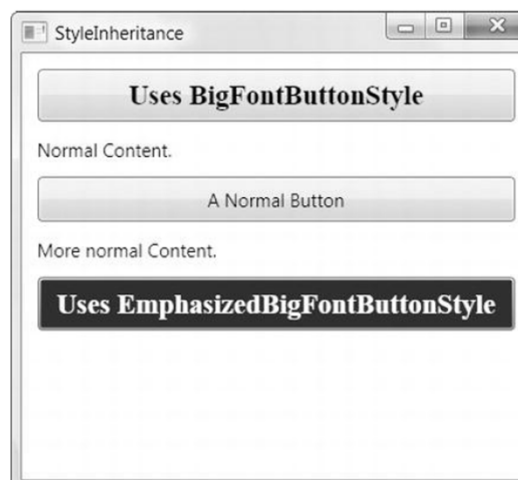
- The first style (`BigFontButtonStyle`) defines **three font properties**.
- The second style (`EmphasizedBigFontButtonStyle`) acquires these aspects from `BigFontButtonStyle` and then **supplements** them with **two more properties** that change the foreground and the background brushes.

```
<Window.Resources>
  <Style x:Key="BigFontButtonStyle">
    <Setter Property="Control.FontFamily" Value="Times New Roman" />
    <Setter Property="Control.FontSize" Value="18" />
    <Setter Property="Control.FontWeight" Value="Bold" />
  </Style>

  <Style x:Key="EmphasizedBigFontButtonStyle"
    BasedOn="{StaticResource BigFontButtonStyle}">
    <Setter Property="Control.Foreground" Value="White" />
    <Setter Property="Control.Background" Value="DarkBlue" />
  </Style>
</Window.Resources>
```

The Many Layers of Styles

- Example



The Many Layers of Styles

- The first style (`BigFontButtonStyle`) defines **three font properties**.
- The second style (`EmphasizedBigFontButtonStyle`) acquires these aspects from `BigFontButtonStyle` and then **supplements** them with **two more properties** that change the foreground and the background brushes.

```
<Window.Resources>
  <Style x:Key="BigFontButtonStyle">
    <Setter Property="Control.FontFamily" Value="Times New Roman" />
    <Setter Property="Control.FontSize" Value="18" />
    <Setter Property="Control.FontWeight" Value="Bold" />
  </Style>

  <Style x:Key="EmphasizedBigFontButtonStyle"
    BasedOn="{StaticResource BigFontButtonStyle}">
    <Setter Property="Control.Foreground" Value="White" />
    <Setter Property="Control.Background" Value="DarkBlue" />
  </Style>
</Window.Resources>
```