Styles

Theming an Application

Styles and Templates enable the user to create a consistent appearance for their application

Style

A style is a collection of DependencyProperty setters that can be applied to multiple objects typically with the goal of providing a consistent appearance. Typically, styles are stored and accessed from ResourceDictionaries. Consider the following piece of XAML

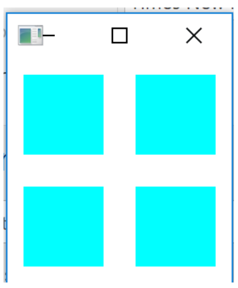
```
<Window x:Class="SimpleStyles.Example1.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        SizeToContent="WidthAndHeight">

    <Window.Resources>
        <Style x:Key="StyleExample">
            <Setter Property="Shape.Fill" Value="Aqua"></Setter>
            <Setter Property="Rectangle.Width" Value="50"></Setter>
            <Setter Property="Rectangle.Height" Value="50"></Setter>
            <Setter Property="Rectangle.Margin" Value="10"></Setter>
        </Style>
    </Window.Resources>

    <StackPanel Orientation="Vertical">
        <StackPanel Orientation="Horizontal">
            <Rectangle Style="{StaticResource StyleExample}"/>
            <Rectangle Style="{StaticResource StyleExample}"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
            <Rectangle Style="{StaticResource StyleExample}"/>
            <Rectangle Style="{StaticResource StyleExample}"/>
        </StackPanel>
    </StackPanel>

</Window>
```

Which renders as follows. Notice our use of the x:Key attribute on the style which enables us to refer to it from it using the StaticResource markup extension in the Rectangles



TargetType

We can use the style TargetType attribute to indicate which types the style can be applied to. This provides us with three benefits.

- Compile-time type checking makes sure we only apply the style to objects of the correct type
- We can omit the type name prefix in the Property attribute of each setter making the code more succinct.
- We can pull in type implicitly by type

Out code from the previous section becomes

```
<Window x:Class="SimpleStyles.Example2.Example2"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  SizeToContent="WidthAndHeight">

    <Window.Resources>
        <Style TargetType="Rectangle">
            <Setter Property="Fill" Value="Aqua"></Setter>
            <Setter Property="Width" Value="50"></Setter>
            <Setter Property="Height" Value="50"></Setter>
            <Setter Property="Margin" Value="10"></Setter>
        </Style>
    </Window.Resources>

    <StackPanel Orientation="Vertical">
        <StackPanel Orientation="Horizontal">
            <Rectangle/>
            <Rectangle/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
            <Rectangle/>
            <Rectangle/>
        </StackPanel>
    </StackPanel>
</Window>
```

Styles with a target type attribute can be classified as either named or typed. A named style has a x:Key attribute whereas a types style does not.

Styles and the resource mechanism

Styles can take advantage of the WPF resource mechanism. Note how in the following example we create a style with different properties on one of the stack panels. The changes are as follows

```xml
<Window x:Class="SimpleStyles.Example3.Example3"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        SizeToContent="WidthAndHeight">

    <Window.Resources>
        <Style TargetType="Rectangle">
            <Setter Property="Fill" Value="Blue"></Setter>
            <Setter Property="Width" Value="50"></Setter>
            <Setter Property="Height" Value="50"></Setter>
            <Setter Property="Margin" Value="10"></Setter>
        </Style>
    </Window.Resources>

    <StackPanel Orientation="Vertical">
        <StackPanel Orientation="Horizontal">
            <Rectangle/>
            <Rectangle/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
            <StackPanel.Resources>
                <Style TargetType="Rectangle">
                    <Setter Property="Fill" Value="Green"></Setter>
                    <Setter Property="Width" Value="50"></Setter>
                    <Setter Property="Height" Value="50"></Setter>
                    <Setter Property="Margin" Value="10"></Setter>
                </Style>
            </StackPanel.Resources>
            <Rectangle/>
            <Rectangle/>
        </StackPanel>
    </StackPanel>
</Window>
```

Extending Styles

We can base one style on another using the BasedOn attribute.

```
<Window x:Class="SimpleStyles.Example5.Example5"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        SizeToContent="WidthAndHeight">

    <Window.Resources>
        <Style x:Key="BaseStyle" TargetType="Rectangle">
            <Setter Property="Fill" Value="Aqua"></Setter>
            <Setter Property="Width" Value="50"></Setter>
            <Setter Property="Height" Value="50"></Setter>
            <Setter Property="Margin" Value="10"></Setter>
        </Style>
        <Style TargetType="Rectangle" BasedOn="{StaticResource BaseStyle}"></Style>
    </Window.Resources>

    <StackPanel Orientation="Vertical">
        <StackPanel Orientation="Horizontal">
            <Rectangle/>
            <Rectangle/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
            <StackPanel.Resources>
                <Style TargetType="Rectangle" BasedOn="{StaticResource BaseStyle}">
                    <Setter Property="Fill" Value="Green"></Setter>
                </Style>
            </StackPanel.Resources>
            <Rectangle/>
            <Rectangle/>
        </StackPanel>
    </StackPanel>
</Window>
```

Uses

1. FrameworkElement.Style
2. FrameworkElement.FocusVisualStyle
3. ItemsControl.ItemContainerStyle
4. FrameworkContentElement.Style
5. FrameworkContentElement.FocusVisualStyle

Questions - Styles

What is a Style?

A style is a collection of DependencyProperty setters that can be applied to multiple objects typically with the goal of providing a consistent appearance

Which feature of WPF enables styles and templates to be reused?

Resources

What are the advantages of using styles?

Extra level of indirection enables changes in one place to affect multiple objects

What are the advantages of using a Style's TargetType attribute?

1. *Compile-time type checking makes sure we only use the style on objects of the correct type*
2. *We can remove the type prefix from the Property attribute in setters making the code more succinct*
3. *We can pull in the style implicitly*

Can a style with a TargetType attribute be applied to a subclass of the TargetType?

It depends if the style is named or typed. Named style can be applied to subclasses whereas typed styles cannot.

How do we make one style extend from another?

Using the BasedOn attribute

If we set the TargetType on a style to TextBlock but don't set an x:Key what is the implicit key?

{x:Type TextBlock}

To which kind of object can a style be applied?

To any element that derives from FrameworkElement and FrameworkContentElement

How does one create a style that can be used anywhere in the application?

Declare it in the root element of your application definition XAML

What happens if multiple triggers impact the same property?

The last one in the trigger collection is the one applied

How does one create a typed style?

Add the TargetType element and omit the x:Key attribute

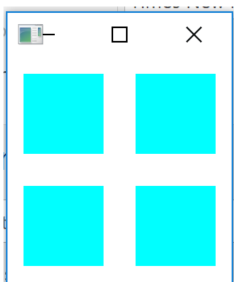
Write code to explicitly access a typed style with type of Button?

<Button Style="{StaticResource{x:Type Button}} ...>

Where are style objects consumed?

1. FrameworkElement.Style
2. FrameworkElement.FocusVisualStyle
3. ItemsControl.ItemContainerStyle
4. FrameworkContentElement.Style
5. FrameworkContentElement.FocusVisualStyle
6. Several tools resource keys for styles that need a default value (dependency properties do not support dynamic resource values as their default value)

Use a basic named style for buttons to create the following



```
<UserControl.Resources>
    <Style x:Key="MyButtonStyle">
        <Style.Setters>
            <Setter Property="Button.Width" Value="50"/>
            <Setter Property="Button.Height" Value="50"/>
            <Setter Property="Button.Margin" Value="10"/>
            <Setter Property="Button.Background" Value="Aqua"/>
        </Style.Setters>
    </Style>
</UserControl.Resources>

<StackPanel Orientation="Vertical">
    <StackPanel Orientation="Horizontal">
        <Button Style="{StaticResource MyButtonStyle}"/>
        <Button Style="{StaticResource MyButtonStyle}"/>
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Button Style="{StaticResource MyButtonStyle}"/>
        <Button Style="{StaticResource MyButtonStyle}"/>
    </StackPanel>
</StackPanel>
```

Change the previous code so the style type is explicit for Button

```
     <Style TargetType="Button">
          <Style.Setters>
              <Setter Property="Width" Value="50"/>
              <Setter Property="Height" Value="50"/>
              <Setter Property="Margin" Value="10"/>
              <Setter Property="Background" Value="Aqua"/>
          </Style.Setters>
      </Style>
      </UserControl.Resources>

      <StackPanel Orientation="Vertical">
          <StackPanel Orientation="Horizontal">
              <Button />
              <Button />
          </StackPanel>
          <StackPanel Orientation="Horizontal">
              <Button />
              <Button />
          </StackPanel>
      </StackPanel>
```

Use the resource mechanism to override the control level style with a stack panel level style in one of the StackPanels

```
      <Style TargetType="Button">
          <Style.Setters>
              <Setter Property="Width" Value="50"/>
              <Setter Property="Height" Value="50"/>
              <Setter Property="Margin" Value="10"/>
              <Setter Property="Background" Value="Aqua"/>
          </Style.Setters>
      </Style>
      </UserControl.Resources>

      <StackPanel Orientation="Vertical">
          <StackPanel Orientation="Horizontal">
              <StackPanel.Resources>
                  <Style TargetType="Button">
                      <Style.Setters>
                          <Setter Property="Width" Value="50"/>
                          <Setter Property="Height" Value="50"/>
                          <Setter Property="Margin" Value="10"/>
                          <Setter Property="Background" Value="Green"/>
                      </Style.Setters>
                  </Style>
              </StackPanel.Resources>
              <Button />
              <Button />
          </StackPanel>
          <StackPanel Orientation="Horizontal">
              <Button />
              <Button />
          </StackPanel>
      </StackPanel>
```

Change the code to base the style in the StackPanel on the style from the UserControl?

```
<UserControl.Resources>
    <Style x:Key="BaseStyle" TargetType="Button">
        <Style.Setters>
            <Setter Property="Width" Value="50"/>
            <Setter Property="Height" Value="50"/>
            <Setter Property="Margin" Value="10"/>
            <Setter Property="Background" Value="Aqua"/>
        </Style.Setters>
    </Style>
    <Style TargetType="Button" BasedOn="{StaticResource BaseStyle}"/>
</UserControl.Resources>

<StackPanel Orientation="Vertical">
    <StackPanel Orientation="Horizontal">
        <StackPanel.Resources>
            <Style BasedOn="{StaticResource BaseStyle}"
                   TargetType="Button">
                <Style.Setters>
                    <Setter Property="Background" Value="Green"/>
                </Style.Setters>
            </Style>
        </StackPanel.Resources>
        <Button />
        <Button />
    </StackPanel>
    <StackPanel Orientation="Horizontal">
        <Button />
        <Button />
    </StackPanel>
</StackPanel>
```

Triggers

Triggers are added to styles in order to modify the look and feel in response to different kind of change. Each style has a triggers collection to which we can add four types of trigger

- (Property)Trigger – act when a `DependencyProperty` changes
- MultiTrigger – property trigger with multiple clauses
- DataTrigger – act when **any kind** of property changes
- MultiDataTrigger - data trigger with multiple clauses
- Event triggers – Invoked when a RoutedEvent is raised

OTHER PLACES WITH TRIGGER COLLECTIONS

In addition to Style, DataTemplates and ControlTemplate also have a Triggers collection which support all the above-mentioned types of trigger. FrameworkElement also has a Triggers collection but it only supports event triggers

(PROPERTY)TRIGGER

Set the value of one or more dependency properties in response to a change in value of another dependency property on the same object.

```
    <Window.Resources>
        <Style TargetType="TextBox">
            <Setter Property="Background" Value="Aquamarine" />
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Background" Value="Red" />
                    <Setter Property="Foreground" Value="White" />
                </Trigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>

    <StackPanel>
        <TextBox>Second TextBox</TextBox>
        <Button x:Name="MyButton">Click Me</Button>
    </StackPanel>
```

DATATRIGGER

A data trigger can be used to Set a dependency property when any bound data changes. The bound data need not be on the same DependencyObject whose DependencyProperties are being updated. The bound data also does not need to be a DependencyProperty (Although as with all Triggers the object being updated must be a dependency property)

```
   <Window.Resources>
        <Style TargetType="TextBox">
            <Setter Property="Background" Value="Aquamarine" />
            <Style.Triggers>
                <DataTrigger Binding="{Binding ElementName=MyButton, Path=IsMouseO-
ver}" Value="True">
                    <Setter  Property="Background" Value="Red" />
                    <Setter  Property="Foreground" Value="DarkOrchid" />
                </DataTrigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>

    <StackPanel>
        <TextBox>Second TextBox</TextBox>
        <Button x:Name="MyButton">Click Me</Button>
    </StackPanel>
```

MultiTrigger/MultiDataTrigger – logical AND

This example uses a MultiTrigger to only apply the setters to the text box that has both focus and a mouse over.

```
<Window.Resources>
    <Style TargetType="TextBox">
        <Setter Property="Background" Value="Aquamarine" />
        <Style.Triggers>
            <MultiTrigger>
                <MultiTrigger.Conditions>
                    <Condition Property="IsMouseOver" Value="True"/>
                    <Condition Property="IsFocused" Value="True"/>
                </MultiTrigger.Conditions>
                <Setter Property="Background" Value="Red" />
            </MultiTrigger>
        </Style.Triggers>
    </Style>
</Window.Resources>

<StackPanel>
    <TextBox>Second TextBox</TextBox>
    <TextBox>Second TextBox</TextBox>
</StackPanel>
```

The following example uses a MultiDataTrigger to achieve a logical AND when the properties we are checking are not DependencyProperties or are not on the object we are targeting with the setters

```
<Window x:Class="Triggers.MultiDataTrigger.MultiDataTriggerWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:viewModels="clr-namespace:SharedResources.ViewModels;assembly=SharedResources"
        mc:Ignorable="d"
        Title="MultiDataTriggerWindow" SizeToContent="WidthAndHeight">
    <StackPanel>
        <StackPanel.Resources>
            <Style TargetType="TextBox">
                <Style.Triggers>
                    <MultiDataTrigger>
                        <MultiDataTrigger.Conditions>
                            <Condition Value="Kenny" Binding="{Binding Path=FirstName}" />
                            <Condition Value="Wilson" Binding="{Binding Path=SecondName}"/>
                        </MultiDataTrigger.Conditions>
                        <Setter Property="Background" Value="LightBlue"></Setter>
                    </MultiDataTrigger>
                </Style.Triggers>
            </Style>
        </StackPanel.Resources>
        <TextBox Text="{Binding Path=FirstName}">
                <TextBox.DataContext>
                    <viewModels:Person FirstName="Kenny" SecondName="Wilson"/>
                </TextBox.DataContext>
            </TextBox>
            <TextBox Text="{Binding Path=FirstName}">
                <TextBox.DataContext>
                    <viewModels:Person FirstName="John" SecondName="Wilson"/>
                </TextBox.DataContext>
            </TextBox>
        </StackPanel>
</Window>
```

Logical OR

We can create a logical OR with triggers by adding multiple triggers with difference source properties and the same setters

Questions -Triggers

What is a Trigger?

Added to styles to conditionally modify DependencyProperty Values

What do Triggers act on?

DependencyProperties

What are the different type of Triggers?

1. *(Property)Trigger respond to changes in DependencyProperty*
2. *MultiTrigger a (Property)Trigger with multiple clauses*
3. *DataTrigger responds to change in any property*
4. *MultiDataTrigger is a DataTrigger with multiple clauses*
5. *EventTrigger – Invoked when a routed event is raised*

When would one use a PropertyTrigger?

When I want to update the value of a dependency property in response to a charge of a Dependency on the same dependency object

When would one use a DataTrigger

1. *When we want to change the value of a DependencyProperty in response to change in a non-dependency property*
2. *When we want to change the value of a DependencyProperty in response to change in a different object from the one we are updating*

What happens if multiple triggers impact the same property?

The last one in the trigger collection is the one applied

How can one achieve logical and with triggers?

By using a MultiTrigger or a MultiDataTrigger

How can one achieve logical or with triggers?

By adding multiple triggers with different conditions and the same setters

Write a trigger to change the background on a TextBlock to DarkBlue and foreground to White when the mouse is over that textblock

```
<Window.Resources>
    <Style TargetType="TextBlock">
        <Style.Triggers>
            <Trigger Property="IsMouseOver" Value="True">
                <Trigger.Setters>
                    <Setter Property="Background" Value="DarkBlue"/>
                    <Setter Property="Foreground" Value="White"/>
                </Trigger.Setters>
            </Trigger>
        </Style.Triggers>
    </Style>
</Window.Resources>


<TextBlock>Hello World</TextBlock>
```

Write a trigger to change the background on a TextBlock to DarkBlue and foreground to White when the mouse is over a Button

```
<Window.Resources>
        <Style TargetType="TextBlock">
            <Style.Triggers>
                <DataTrigger Binding="{Binding ElementName=MyButton,
                    Path=IsMouseOver}" Value="True">
                    <Setter Property="Background" Value="DarkBlue"/>
                    <Setter Property="Foreground" Value="White"/>
                </DataTrigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>

    <StackPanel>
        <TextBlock Width="200" Height="30">Hello World</TextBlock>
        <Button x:Name="MyButton" Width="200" Height="30">ClickMe</Button>
    </StackPanel>
```

Write a trigger to change the background on a TextBlock to Red if it has both focus and the mouse is over it

```
<Window.Resources>
    <Style TargetType="TextBox">
        <Style.Triggers>
            <MultiTrigger>
                <MultiTrigger.Conditions>
                    <Condition Property="IsMouseOver" Value="True"/>
                    <Condition Property="IsFocused" Value="True"/>
                </MultiTrigger.Conditions>
                <Setter Property="Background" Value="Red"></Setter>
            </MultiTrigger>
        </Style.Triggers>
    </Style>
</Window.Resources>


<StackPanel>
    <TextBox>First Text Box</TextBox>
    <TextBox>Second Text Box</TextBox>
</StackPanel>
```

Write a trigger to change the background on a TextBlock to Red if it has both focus and the mouse is over it

ToDo

Dependency Properties and Dynamic resources as default value

“One reason ToolBar uses ResourceKey property instead of Style properties is that depedency properties do not support dynamic resource values as their default value. ItemsControl can get away with giving ItemContainerStyle a default value of null because the default style for the item container is always the same. ToolBar, however, requires different default styles, depending on the theme”

What does the above mean?