Controls

Separating the look from the behaviour

Overview

The killer feature of WPF controls is that the procedural behaviour code is completely separated from its visual tree. Every Control has a `Template` dependency property of type `ControlTemplate`. A control template enables the consumer of a control to replace its visual tree with a completely new and arbitrarily complex tree of visuals while keeping the core behaviour of the control intact. Whenever a Control is instantiated its ControlTemplate is used to generate a tree of visuals that will be used in rendering. The template acts as a blueprint that tells WPF how to create the visual elements needed to render the Control. The content of a ControlTemplate is of type `VisualTree`.

All FrameworkElements in a control template have a relationship back to the control being templated. This relationship is known as the templated parent and is represented by the property

Questions - Overview

What is the killer feature of WPF controls?

The procedural behaviour code is completely separated from its visual tree

What defines a control?

Its behaviour

Is a WPF control directly responsible for its visuals?

No. It delegates to an instance of ControlTemplate

The template has complete control over the rendering of the control

How is the ControlTemplate specified?

By setting a DependencyProperty called Template.

What is the benefit of this approach?

No need to write a custom control to change the appearance of a control.

Consumer can replace look with a new and arbitrarily complex tree of visual objects.

Core behaviour remains intact

How does a ControlTemplate work?

When a control is instantiated its ControlTemplate is used to create a visual tree of elements

The Visual tree of elements render the control

Is the ControlTemplate the visual tree?

No. The ControlTemplate is a blueprint used used to generate the visual tree.

It is not the visual tree.

What is the type of the content of the ControlTemplate?

VisualTree

When do you need to write a custom control?

You need to provide a new behaviour not provided by existing controls

All FrameworkElements in a control template have a relationship back to the control being templated. This relationship is known as?

The templated parent

In what way can one change the appearance of a control?

1. *Setting properties directly such as foreground*
2. *With styles*
3. *With templates*
4. *Custom controls we can place graphics*

Are all UI elements in WPF controls?

No

What is the base class for the UI elements that don't have behaviour?

FrameworkElement

Templating a standard control

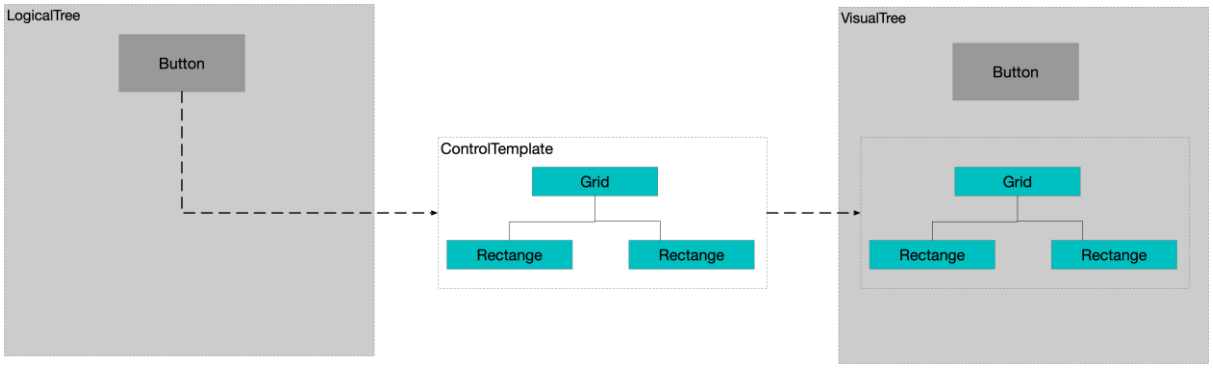
Creating the ControlTemplate

Listing 1Creating a ControlTemplate

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
  </Grid>
</ControlTemplate>
```

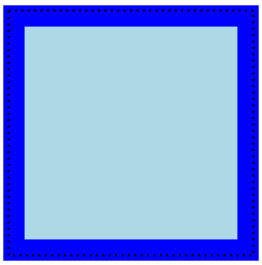
The ControlTemplate is not the same thing as the rendered visuals in the visual tree. It is a blueprint which tells WPF how to create the VisualTree as shown in the diagram below

Figure 1Creating a ControlTemplate



If we add a button whose Template is set to our ControlTemplate it is rendered as follows.

RENDERED 1CREATING A CONTROLTEMPLATE



Triggers and visual state

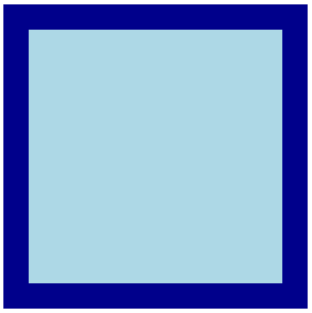
Listing 2 Triggers and Visual State

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
  </Grid>
  <ControlTemplate.Triggers>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter TargetName="Rectangle1" Property="Fill" Value="DarkBlue"/>
    </Trigger>
    <Trigger Property="IsPressed" Value="True">
      <Setter Property="RenderTransform">
        <Setter.Value>
          <ScaleTransform ScaleX=".9" ScaleY=".9"/>
        </Setter.Value>
      </Setter>
      <Setter Property="RenderTransformOrigin" Value=".5,.5"/>
    </Trigger>
  </ControlTemplate.Triggers>
</ControlTemplate>
```

Note how we use a special `TargetName` attribute on the triggers setter to target a specific visual element in the control template. This attribute is only used within Templates. A visual element inside a template can take an `x:Name` attribute. This attribute can only be used to access the element from the templates triggers which is somewhat different from its use on elements that form part of the logical tree. The reason being that any particular part of a template could correspond to multiple visuals in the visual tree if the template is applied multiple times.

RENDERED 2TRIGGERS AND VISUALSTATE

Now when the mouse is over the button the fill on the outer rectangle changes to a dark blue colour.



Templated Parent

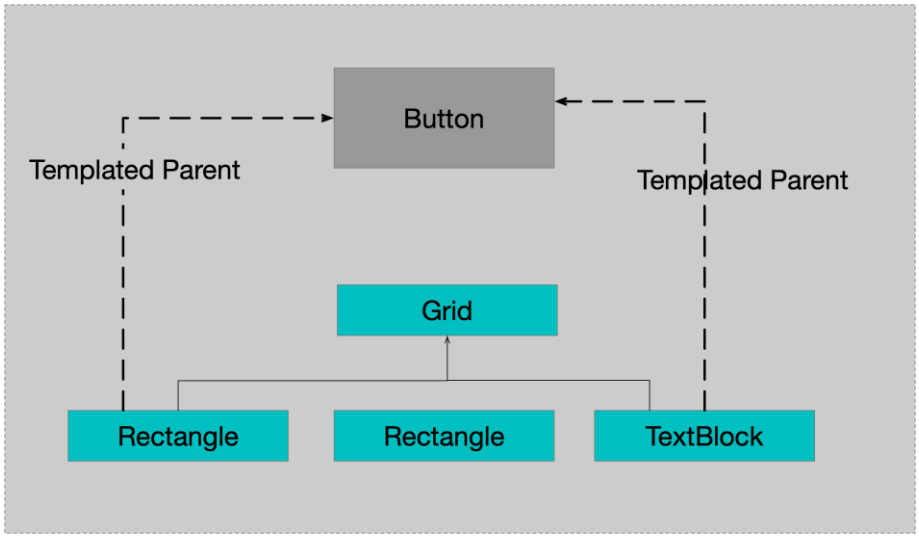
A templated control is considered the templated parent of all the visual elements in its visual tree. Every FrameworkElement has a property called TemplatedParent which enables template elements to access the templated parent. FrameworkElement has a TemplatedParent property that refers to the templated parent.

Listing 3 Templated Parent

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
   <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
     Text="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=Content}"/>
</Grid>
```

Figure 2Templated Parent

Notice how each rectangle can reference its templated parent



We can use a special markup extension called TemplateBinding. This makes the xaml more concise, however it has limitations when compared to the full blown Binding markup extension. The limitations are as follows.

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
   <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
     Text="{TemplateBinding Property=Content}" />

</Grid>
```

Respecting Content

A `Button` is a `ContentControl` which means it should render a single piece of content. The manner in which we implemented the ControlTemplate in the previous section meant that the window would always render text correctly in a TextBlock, however any other type of content would not be correctly rendered. We now consider how to create a ControlTemplate for our Button that correctly renders its content. The most common way to render Content in a ContentControl is via an instance of the `ContentPresenter` class. ContentPresenter is a FrameworkElement designed for rendering heterogenous content. Every WPF ContentControl uses an instance of ContentPresenter in its default ControlTemplate.

Listing 4Respecting Content

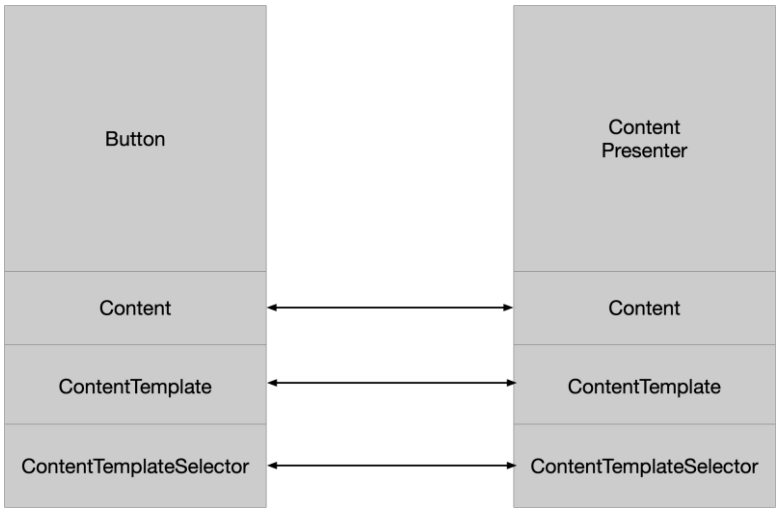
```
<Grid>
  <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
  <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
  <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"
    Content="{TemplateBinding Property=Content}"/>
</Grid>
```

The ContentPresenter has three core DependencyProperties.

- Content
- ContentTemplate
- ContentTemplateSelector

Whenever a ContentPresenter is inside the ControlTemplate of a ContentControl these three properties automatically take their values from the properties of the same name in the parent ContentControl

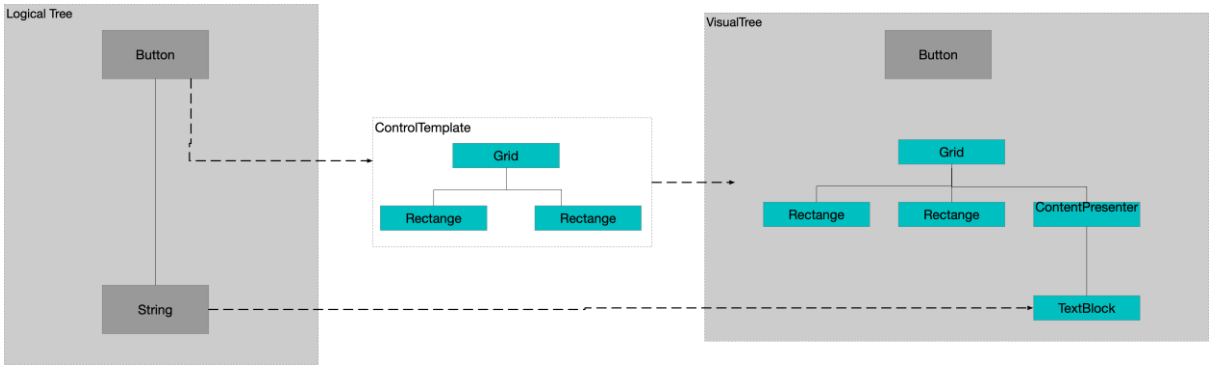
Figure 3 ContentPresenter Properties



Let us add some text to our Button as content and see how it renders

```
<Button Content="Click Me"/>
```

Figure 4ContentPresenter and string content



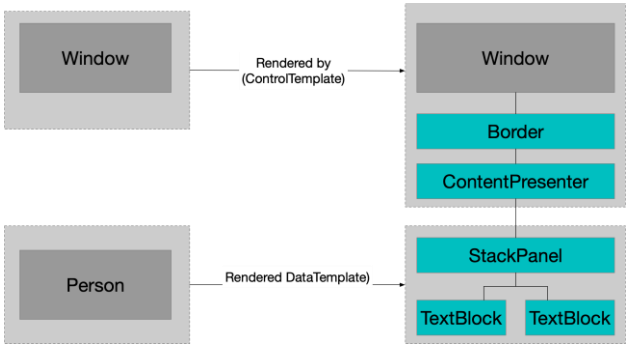
How does the ContentPresenter know to convert the Button content string to a TextBlock and set its TextProperty to be the content string? This is one of the rules of the WPF content model which we can summarise as follows.

1. If the content is a UI Element render it as is
1. If the content is a string render it in a TextBlock
2. If the content is a.NET type which has a data template associated with it then use the data template to render it.
3. Otherwise render the standard .NET objects class name in a TextBlock

DataTemplates and ContentControls

In this example the window's content is a complex object which we render via a DataTemplate.

Figure 5Control and Data Templates

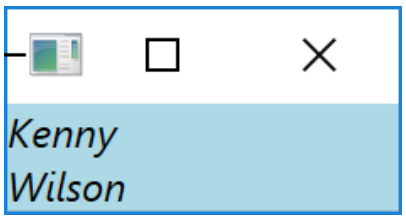


Listing 5Content and Data Templates

```
<Window.Resources>
  <DataTemplate DataType="{x:Type viewModels:Person}">
    <StackPanel Orientation="Vertical">
      <TextBlock Text="{Binding FirstName}"></TextBlock>
      <TextBlock Text="{Binding SecondName}"></TextBlock>
    </StackPanel>
  </DataTemplate>
</Window.Resources>

<Window.Template>
  <ControlTemplate TargetType="{x:Type templatingStandardControls:_05ContentAndDataBinding}">
    <Border Background="LightBlue">
      <ContentPresenter></ContentPresenter>
    </Border>
  </ControlTemplate>
</Window.Template>
<viewModels:Person FirstName="Kenny" SecondName="Wilson"/>
```

LISTING 6RENDERED WINDOW



Content Control's ContentTemplate

We have already seen that each control has a ControlTemplate that defines the visual tree of that Control. In addition every ContentControl also has a ContentTemplate which defines how its single piece of content is rendered. While the Control.Template dependency property is of type ControlTemplate the ContentTemplate is of type DataTemplate.

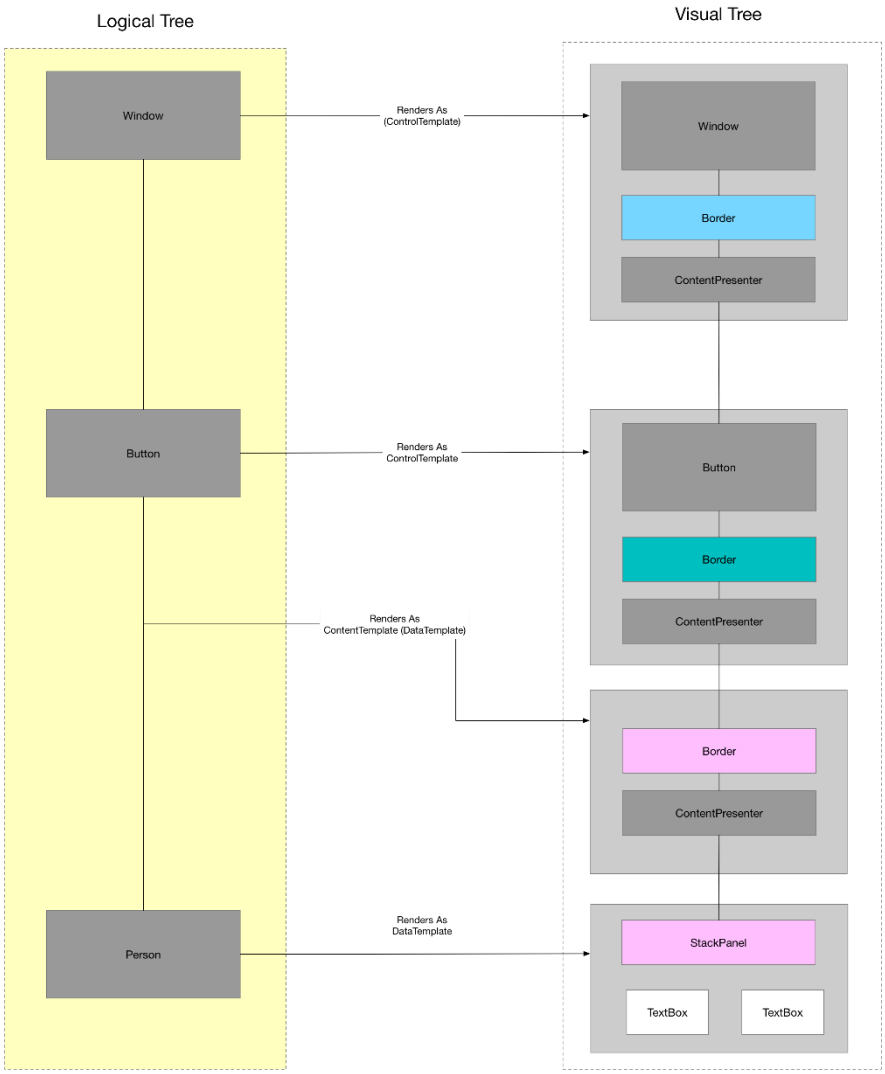
DataTemplates and ControlTemplates while similar (both extend the same base class) are intended for different purposes. A control template defines the look of the whole control and tends to be somewhat complex. A data template defines the look of a piece of content within a content control and tends to be simpler. For instance consider the scenario where I want to render a non UIElement object as the content of a button.

The following example is a little contrived but it uses two control templates (one for window and one for button), a content template (for button) and a data template to render the person object. In reality one might use a content template to provide some simpler customisation rather than writing the whole control template from scratch)

Listing 7ContentTemplate

```
<Window.Resources>
  <DataTemplate DataType="{x:Type viewModels:Person}">
    <StackPanel>
      <TextBlock Background="White" Margin="1" Padding="5" Text="{Binding FirstName}"/>
      <TextBlock Background="White" Margin="1" Padding="5" Text="{Binding SecondName}"/>
    </StackPanel>
   </DataTemplate>
     <Style TargetType="Button">
       <Setter Property="Margin" Value="5"/>
       <Setter Property="Padding" Value="5"/>
       <Setter Property="Template" >
       <Setter.Value>
         <ControlTemplate TargetType="Button">
           <Border Background="LightSeaGreen" Padding="5">
             <ContentPresenter/>
           </Border>
         </ControlTemplate>
       </Setter.Value>
     </Setter>
   </Style>
 </Window.Resources>
 <Window.Template>
 <ControlTemplate
TargetType="{x:Type templatingStandardControls:_06ContentControlAndDataTemplates}">
  <Border Background="{TemplateBinding Background}">
    <ContentPresenter></ContentPresenter>
  </Border>
</ControlTemplate>
</Window.Template>
  <Button Content="{viewModels:PersonMarkup FirstName=Kenny, SecondName=Wilson}">
    <Button.ContentTemplate>
      <DataTemplate>
        <Border Padding="5" Margin="5" Background="LightPink">
          <ContentPresenter Content="{Binding}"></ContentPresenter>
            </Border>
            </DataTemplate>
        </Button.ContentTemplate>
  </Button>
```

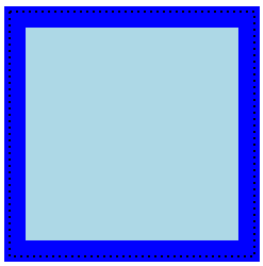
Figure 6ContentTemplate



Questions – Templating a standard control

Creating a ControlTemplate

Add a ControlTemplate to make a button render as follows



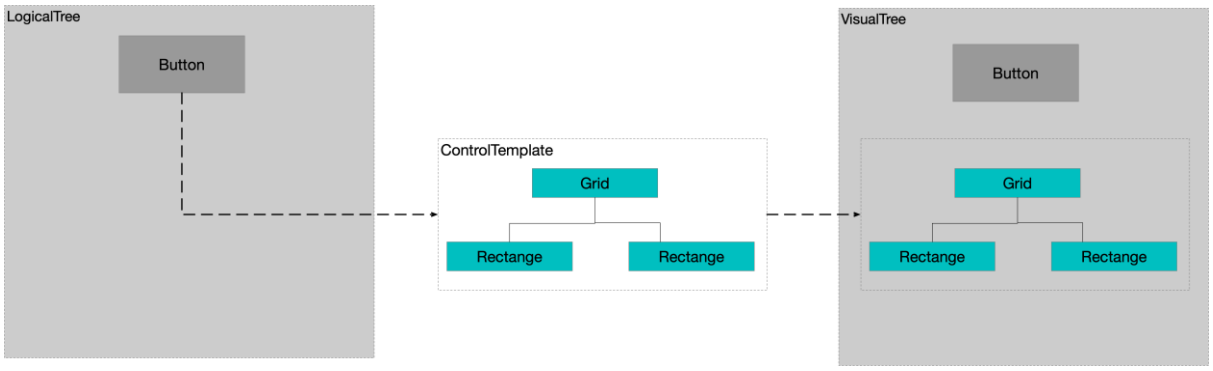
Answer

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
  </Grid>
</ControlTemplate>
```

Show how the following piece of xaml is used to create the look of the button?

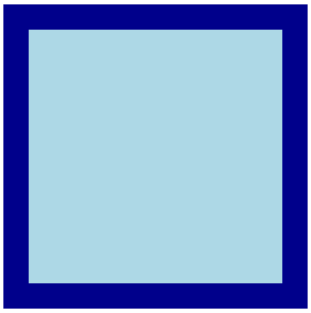
```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
  </Grid>
</ControlTemplate>
```

Answer



Triggers and Visual State

Add two triggers to the previous questions XAML to when the mouse hovers over the button the outer rectangle changes to dark blue and when the Button is clicked it shrinks to 90% of its size



Answer

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
  </Grid>
  <ControlTemplate.Triggers>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter TargetName="Rectangle1" Property="Fill" Value="DarkBlue"/>
    </Trigger>
    <Trigger Property="IsPressed" Value="True">
      <Setter Property="RenderTransform">
        <Setter.Value>
          <ScaleTransform ScaleX=".9" ScaleY=".9"/>
        </Setter.Value>
      </Setter>
      <Setter Property="RenderTransformOrigin" Value=".5,.5"/>
    </Trigger>
  </ControlTemplate.Triggers>
</ControlTemplate>
```

How do Triggers in a ControlTemplate target specific visual elements in the control template?

The TargetName attribute on the trigger's setter

How is the x:Name attribute of a visual element in the control template used?

It can only be used to access the element from the templates trigger

Why?

The template part could correspond to multiple visuals in the visual tree if the template is applied many times

TEMPLATED PARENT

What is the template parent?

A templated control is the template patent of all visual elements in its visual tree

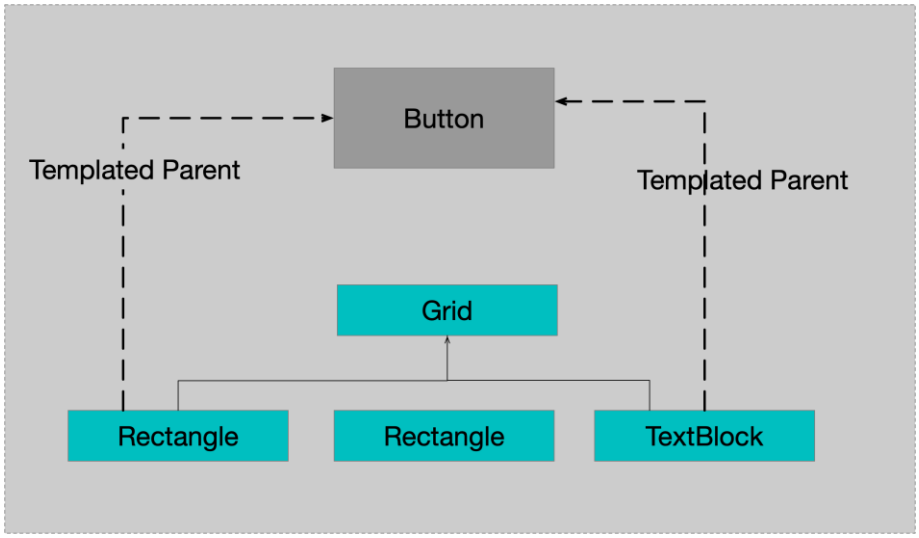
How is the templated parent accessed?

The FrameworkElement.TemplaredParent dependency property

Draw a diagram showing the TemplatedParent for this code

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
   <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
     Text="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=Content}"/>
</Grid>
```

Answer



When binding to a templated parent what special syntax does XAML provide?

The TemplateBinding MarkupExtension

Show two ways of binding the text boxes Content to the Content of the templated patent button in the following Xaml

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
   <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
</Grid>
```

Answer1

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
   <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
     Text="{Binding RelativeSource={RelativeSource TemplatedParent}, Path=Content}"/>
</Grid>
```

Answer 2

```
<ControlTemplate TargetType="Button">
  <Grid>
    <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
    <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
   <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
     Text="{TemplateBinding Property=Content}" />

</Grid>
```

Respecting Content

What is a ContentControl?

A control that renders a single piece of Content

What is the most common way to render Content in a ContentControl?

Via an instance of the ContentPresenter class

What is the ContentPresenter?

A FrameworkElement designed for rendering heterogenous content

Where is ContentPresenter used?

Every WPF ContentControl uses ContentPresenter in its default ControlTemplate

Add code to the following so the control template can render any content

```
<Grid>
  <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
  <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
</Grid>
```

Answer

```
<Grid>
  <Rectangle x:Name="Rectangle1" Width="120" Height="120" Fill="Blue"/>
  <Rectangle x:Name="Rectangle2" Width="100" Height="100" Fill="LightBlue"/>
  <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"
    Content="{TemplateBinding Property=Content}"/>
</Grid>
```

What are the three core DependencyProperties of ContentPresenter?

Content

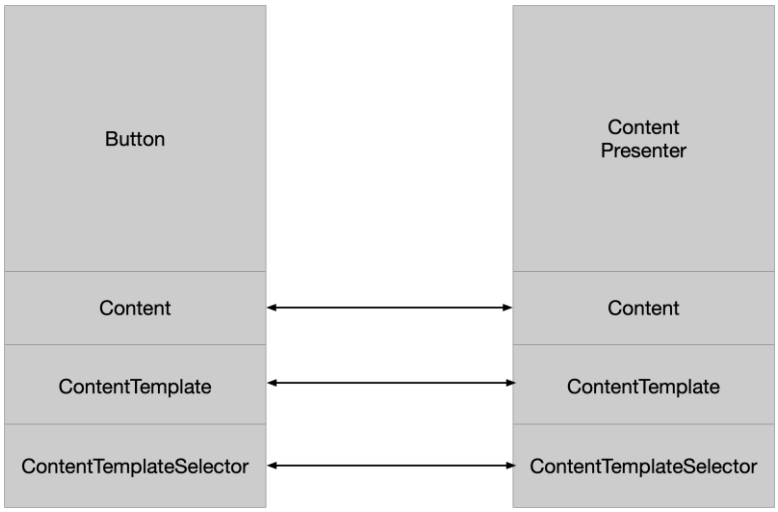
ContentTemplate

ContentTemplateSelector

What shortcutting mechanism exists for these properties?

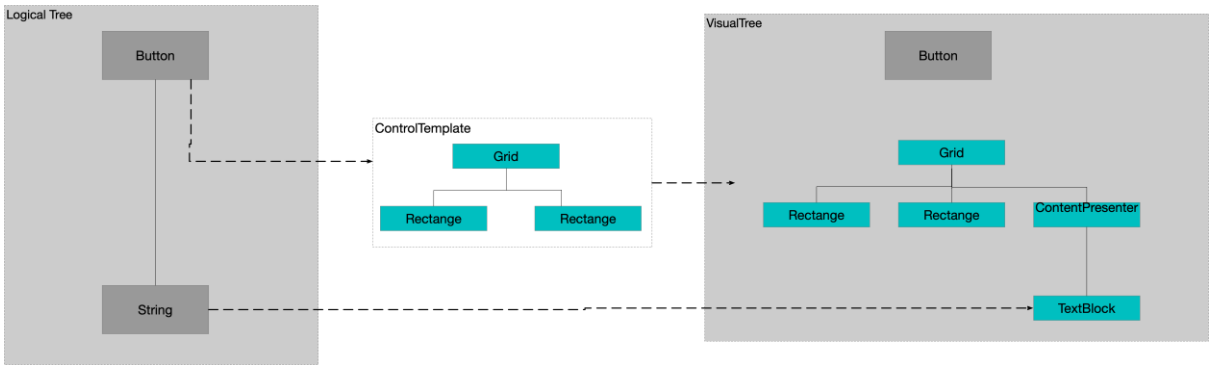
Whenever a ContentPresenter is inside the ControlTemplate of a ContentControl these three properties automatically take their values from the properties of the same name in the parent ContentControl

Show how this mapping looks?



If the Button has text content how is it rendered.?

ToDo: Fix the control template in diagram to add the ContentPresenter



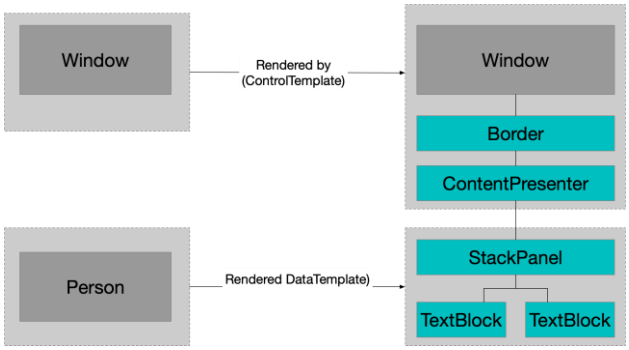
How does the ContentPresenter know to convert the Buttons content string to a TextBlock and set its TextProperty to be the content string?

The rules of the WPF Content model

What are the rules

1. *If the content is a UI Element then render it as is*
2. *If the content is a string render it in a textblock*
3. *If the content is a .NET type which has a data template associated with it then use the date template to render it*
4. *Otherwise render the standard .net objects class name in a TextBlock*

Given the following diagram write the Xaml




```
<viewModels:Person FirstName="Kenny" SecondName="Wilson"/>
```

Answer

```
<Window.Resources>
  <DataTemplate DataType="{x:Type viewModels:Person}">
    <StackPanel Orientation="Vertical">
      <TextBlock Text="{Binding FirstName}"></TextBlock>
      <TextBlock Text="{Binding SecondName}"></TextBlock>
    </StackPanel>
  </DataTemplate>
</Window.Resources>

<Window.Template>
  <ControlTemplate TargetType="{x:Type templatingStandardControls:_05ContentAndDataBinding}">
    <Border Background="LightBlue">
      <ContentPresenter></ContentPresenter>
    </Border>
  </ControlTemplate>
</Window.Template>
<viewModels:Person FirstName="Kenny" SecondName="Wilson"/>
```

CONTENT CONTROL'S CONTENTTEMPLATE

What is a ContentControl's ContentTemplate

It defines how a single piece of content is rendered

How does ContentTemplate differ from Template?

Control.Template is of type ControlTemplate

ContentControl.ContentTemplate is of type DataTemplate

Compare and contract Template and ContentTemplate?

Template is a ControlTemplate and defines the rendering of the control as a whole.
ContentTemplate is a data template which renders just the content itself.

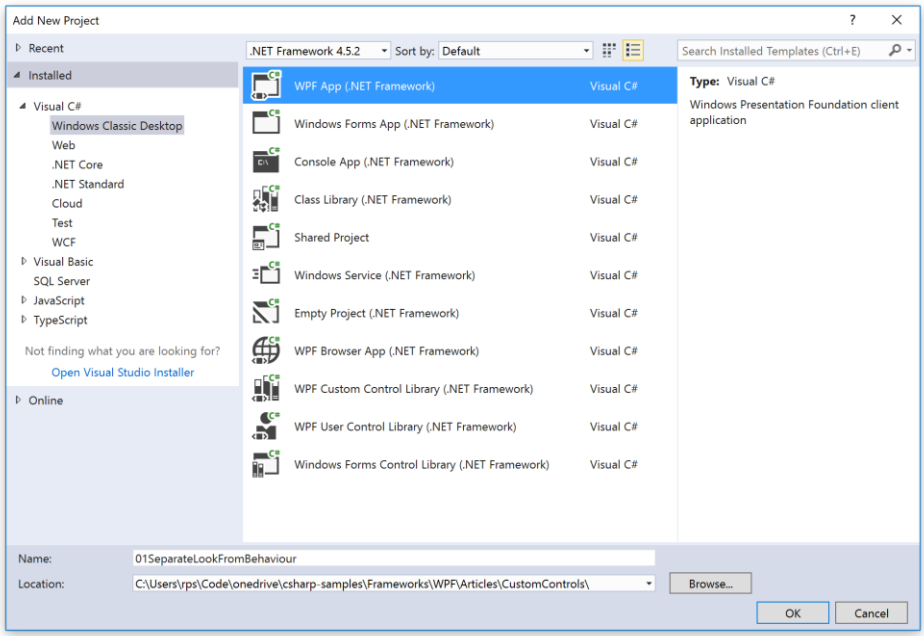
Creating a Custom Control with control template

In this section we will show how to create a custom control. The control in question is a very simple modeless dialogue box. When made visible the dialogue box simply sits on top of other content rather than the traditional model separate dialogue that comes for free with WPF. Although simple this control highlights the following points.

1. Separation of look and feel from behaviour using a control template
2. How do define the contract between the control and its template

1. Add a project to the solution of type WPF App (.NET Framework)

We call our project 01SeparateLookFromBehaviour



2. Add a MyCustomControl class that extends Control

```
class MyCustomControl : Control
{
}
```

3. Add a static constructor with the following boilerplate

```
class MyCustomControl : Control
{
    static MyCustomControl()
    {
        DefaultStyleKeyProperty.OverrideMetadata(typeof(MyCustomControl),
            new FrameworkPropertyMetadata(typeof(MyCustomControl)));
    }
}
```

4. Add our newly created control as our windows content

When we creates our project visual studio adcded a Window1 window for us. We can use this to test our new control by adding it to the window

```
<Window x:Class="SeparateLookFromBehaviour.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:separateLookFromBehaviour="clr-namespace:SeparateLookFromBehaviour"
        Title="MainWindow" Height="350" Width="525">

    <separateLookFromBehaviour:MyCustomControl/>
</Window>
```

Because we have not created any template to defne the look our control is essentially lookless.

5. Add a very basic look

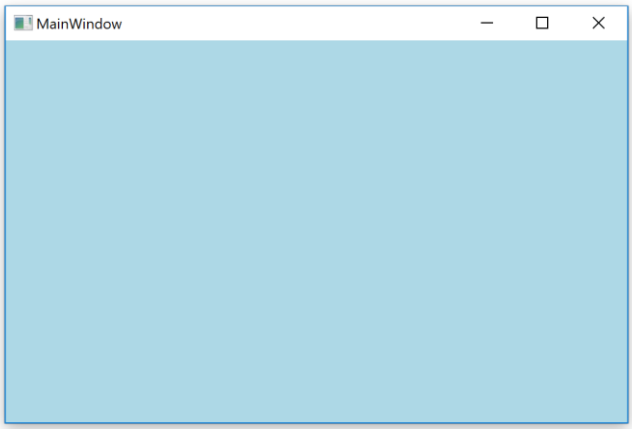
Now we can go ahead and add our default control template that defines the look of our control. Move to Themes/Generic.xaml and add the following

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:separateLookFromBehaviour="clr-namespace:SeparateLookFromBehaviour">

    <Style TargetType="separateLookFromBehaviour:MyCustomControl">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="separateLookFromBehaviour:MyCus-
tomControl">
                    <Border Background="LightBlue"></Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>

</ResourceDictionary>
```

We have themed our control. If we compile and go back to our window it looks as follows.



Define a contract between the control and its template

For all but the simplest of controls we need to specify on some level that the template must contains certain elements. We do this via TemplatePart attributes on the custom control.

1. Add a TemplatePart attribute to the custom control

```
[TemplatePart(Name = "PART_Button",Type = typeof(Button))]
class MyCustomControl : Control
{
    static MyCustomControl()
    {
        DefaultStyleKeyProperty.OverrideMetadata(typeof(MyCustomControl),
            new FrameworkPropertyMetadata(typeof(MyCustomControl)));
    }

    public override void OnApplyTemplate()
    {
        _button = GetTemplateChild("PART_Button") as Button;

        if (_button != null)
        {
            _button.Click += (sender, args) =>
            {


            };
        }
    }

    private Button _button;
}
```

2. Add the required part to the template

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:separateLookFromBehaviour=
                    "clr-namespace:SeparateLookFromBehaviour">

    <Style TargetType="separateLookFromBehaviour:MyCustomControl">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate
                    TargetType="separateLookFromBehaviour:MyCustomControl">
                    <Border Background="LightBlue">
                         <Button Name="PART_Button">Click Me</Button>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>

</ResourceDictionary>
```

Now make sure you can see your handler on the button being hit

Add a Dependency Property

1. Add a static DependencyProperty boilerplate as follows

```
using System.Windows;
using System.Windows.Controls;

namespace SeparateLookFromBehaviour
{
    [TemplatePart(Name = "PART_Button",Type = typeof(Button))]
    class MyCustomControl : Control
    {
        public static DependencyProperty MyValueProperty =
            DependencyProperty.Register("MyValue", typeof(double),
                typeof(MyCustomControl),
                new FrameworkPropertyMetadata(OnValueChanged));

        private static void OnValueChanged(DependencyObject dependencyObject,
            DependencyPropertyChangedEventArgs args)
        {
            var myCustomControl = dependencyObject as MyCustomControl;
            if (myCustomControl?._button != null)
                myCustomControl._button.Content = args.NewValue;
        }

        public double MyValue
        {
            get => (double)GetValue(MyValueProperty);
            set => SetValue(MyValueProperty,value);
        }

        static MyCustomControl()
        {
            DefaultStyleKeyProperty.OverrideMetadata(typeof(MyCustomControl),
                new FrameworkPropertyMetadata(typeof(MyCustomControl)));
        }

        public override void OnApplyTemplate()
        {
            _button = GetTemplateChild("PART_Button") as Button;

            if (_button != null)
            {
                 _button.Content = MyValue;

                _button.Click += (sender, args) =>
                {
                     MyValue = MyValue * 1.1;
                };
            }
        }

        private Button _button;
    }
}
```

2. Create a simple view model

```
    public class ViewModel : INotifyPropertyChanged
    {
        private double _someValue;

        public double SomeValue
        {
            get => _someValue;
            set
            {
                _someValue = value;
                PropertyChanged?.Invoke(this, new PropertyChangedEven-
tArgs(nameof(SomeValue)));
            }
        }
        public event PropertyChangedEventHandler PropertyChanged;
    }
```

3. Set the ViewModel on the View and bind it to our DP

```
<Window x:Class="SeparateLookFromBehaviour.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:separateLookFromBehaviour="clr-namespace:SeparateLookFromBehaviour"
        Title="MainWindow" Height="350" Width="525">

    <Window.DataContext>
        <separateLookFromBehaviour:ViewModel SomeValue="100"/>
    </Window.DataContext>

    <separateLookFromBehaviour:MyCustomControl MyValue="{Binding SomeValue}"/>
</Window>
```

Write code to do the following?

Questions – Creating a custom control

How do we create a custom control?

Subclass Control

Add static constructor and call DefaultStyleKeyMetaData.OverrideMetadata

How do we create the default ControlTemplate for our new control

Add it to a style in Themes/Generic.xaml

How do specify in the control that any template must specify certain elements?

Via TemplatePart class level attributes

How do we access these elements of the template from the control?

Override OnApplyTemplate and call GetTemplateChild

How do we expose state to user of our new control?

By adding dependencyproperties

How do we access changes to the value of the DP's source value?

By registering handler in the DependencyProperty register method

Control List

1. ContentControl – Constrained to contain single item
    a. Button –
        i. Toggle Button
        ii. CheckBox
        iii. Radio Button
2. Simple Containers

    a. Label
    b. Tooltip
3. Headered Containers
    a. GroupBox
    b. Expander
4. Items Control
    a. Selectors
        i. Combo Box
        ii. List Box
        iii. List View
5. Menu
6. Range Control
    a. Progress Bar
    b. Slider
7. Text Controls
    a. TextBox
    b. RickTextBox
    c.