Controls

Separating the look from the behaviour

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 know as the templated parent and is represented by the property

## Templating a standard control

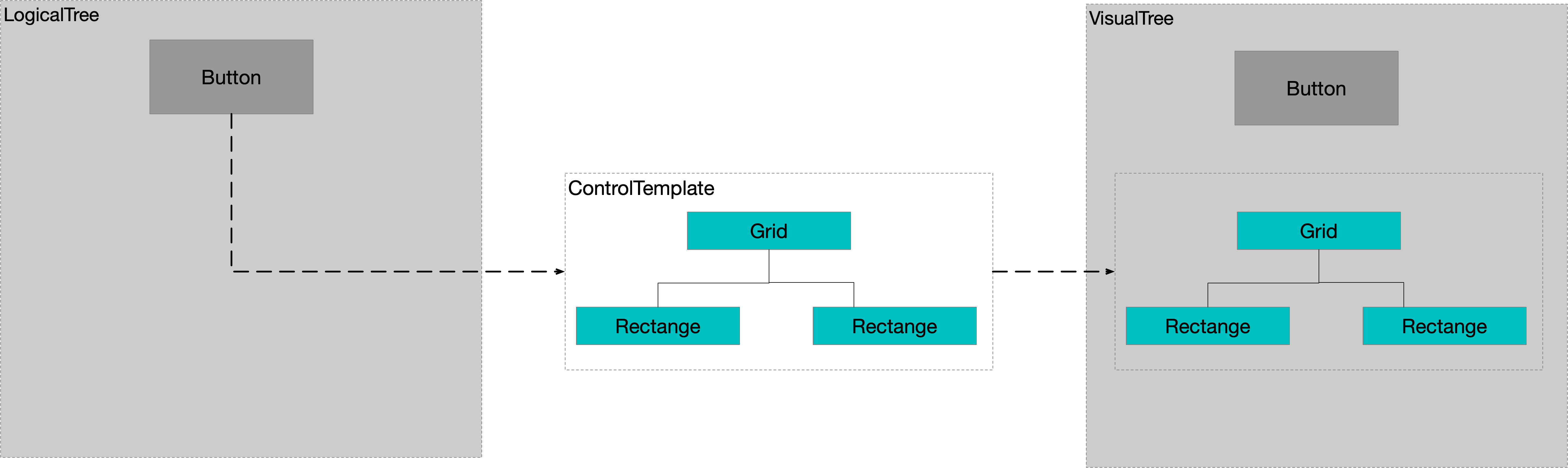
Creating the ControlTemplate

Listing 1Creating a ControlTemplate



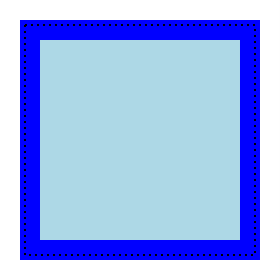
The ControlTemplate is not the same thing at the actually rendered visuals in the visual tree. It is a blueprint which tells WPF how to create the VisualTree as shown in the diagram below

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

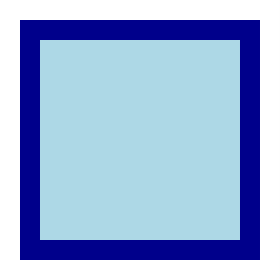
Listing 2 Triggers and Visual State



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

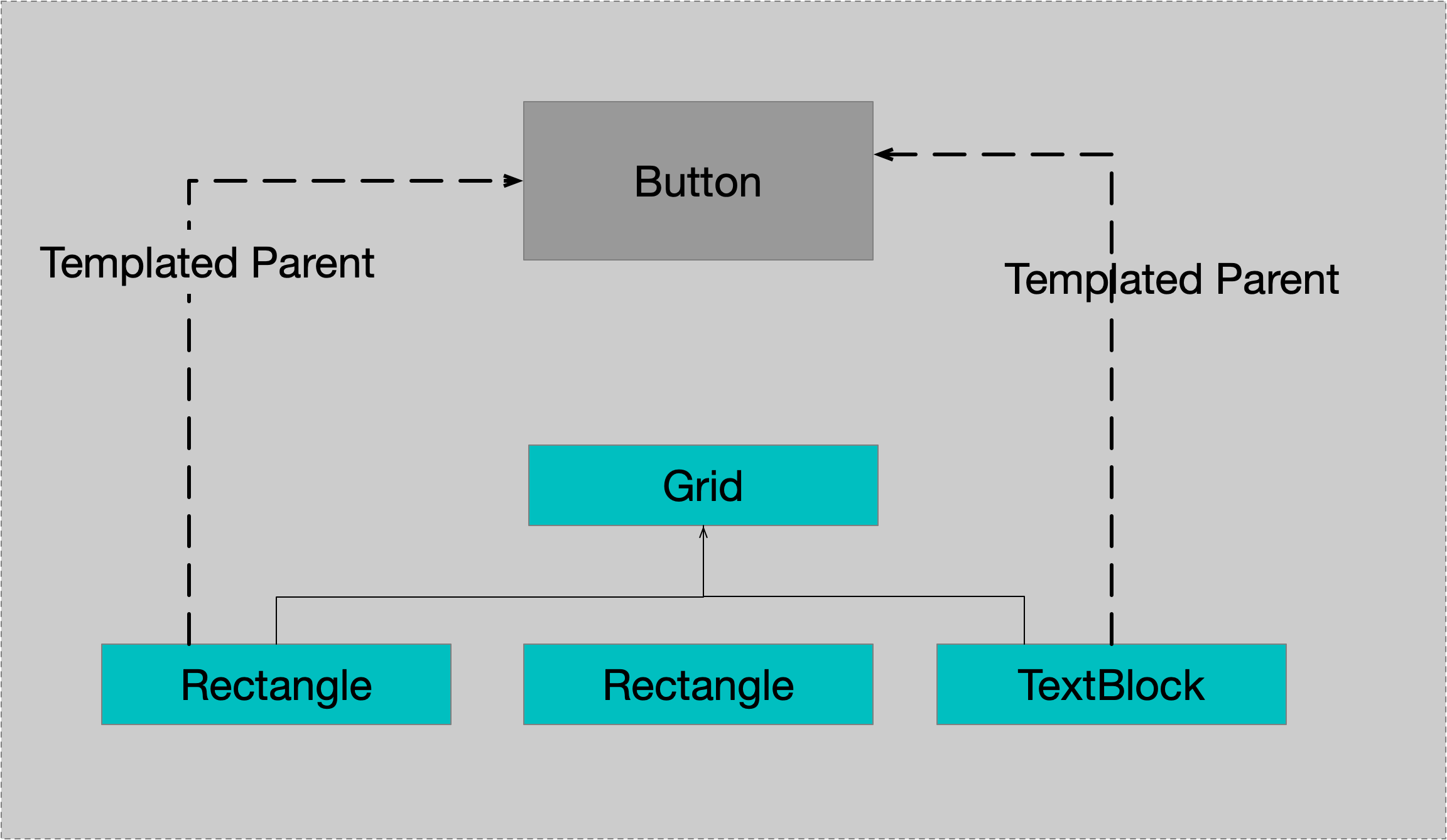
A templated control is considered the templated parent of all the visual elements in its visual tree. Every FrameworkElement has a property called TemplatedParent which enables template elements to access the templated parent. FrameworkElement has a TemplatedParent property that refers to the templated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

Listing 3 Templated Parent



Figure 2Templatd Parent

Notice how each rectange 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.



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

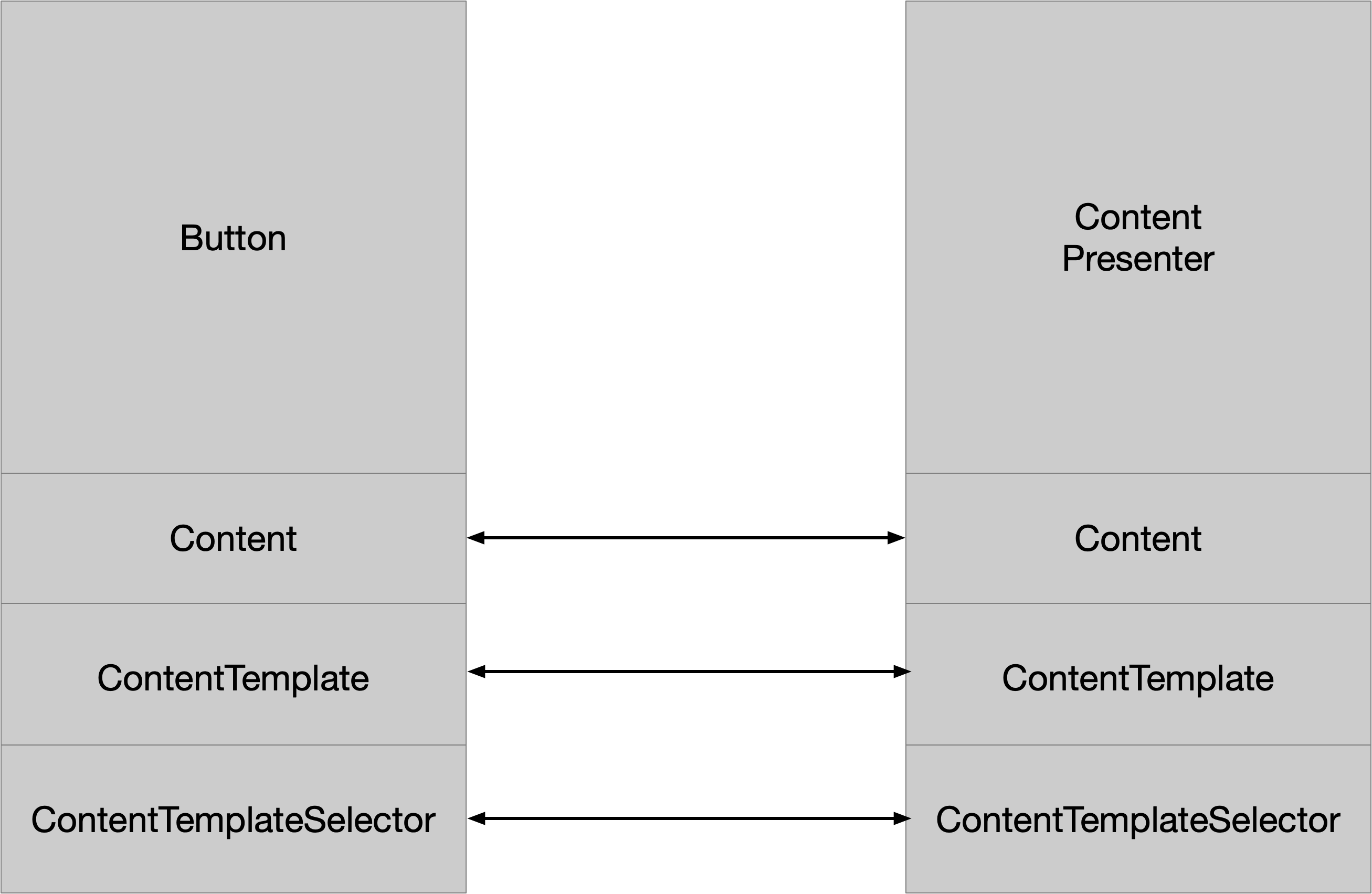


The ContentPresenter has three core DependencyProperties.

* 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

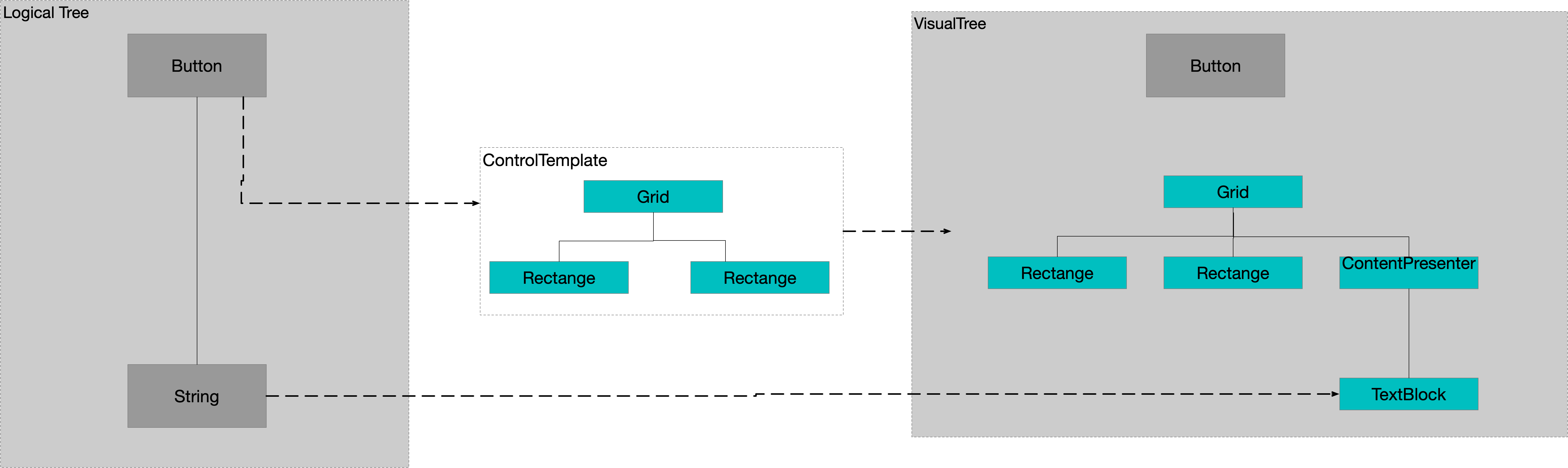
Figure 3ContentPresenter Properties



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



Figure 4ContentPresenter and string content



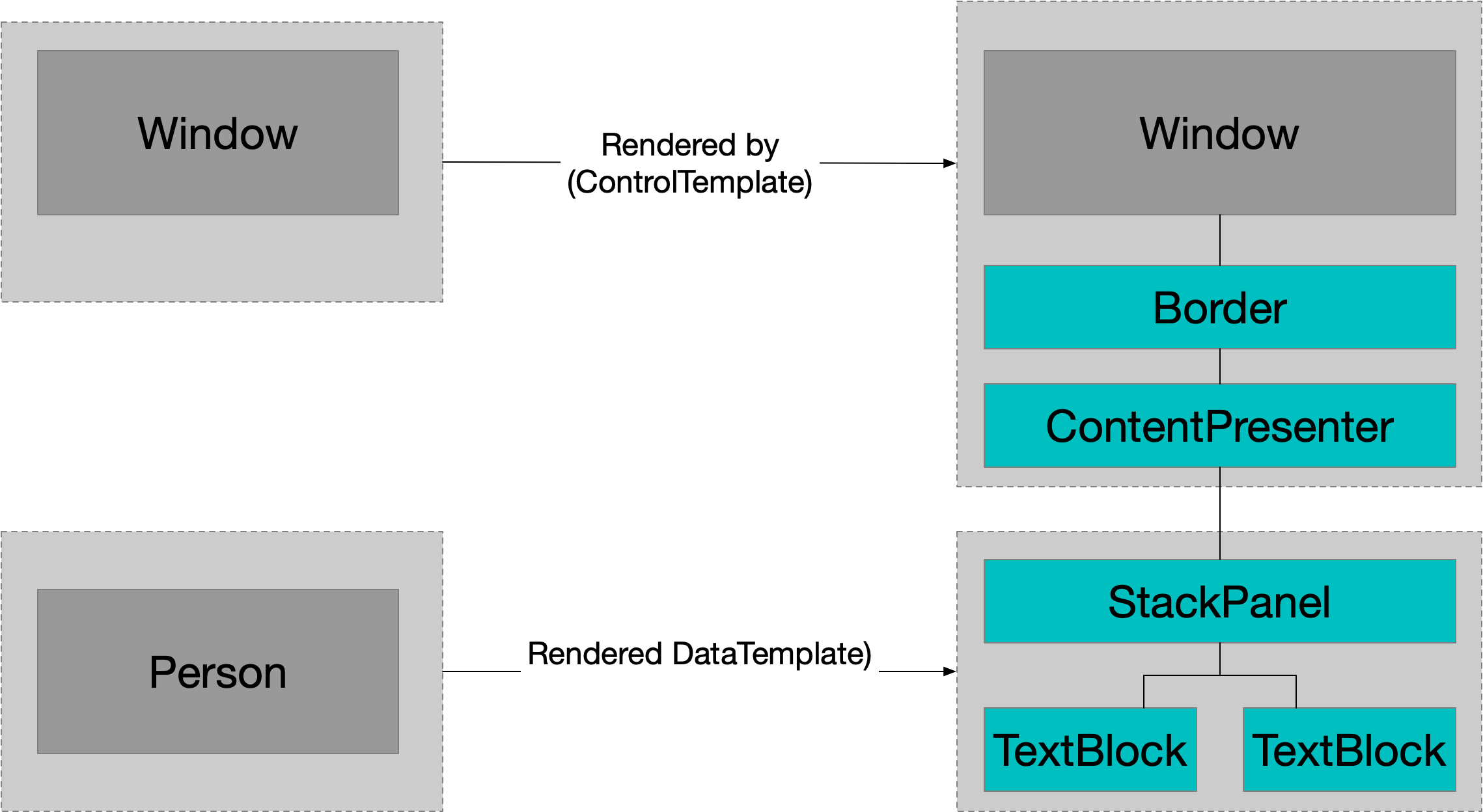
How does the ContentPresenter know to convert the Button content string to a TextBlock and set its TextProperty to be the content string? This is one of the rules of the WPF content model which we can summarise as follows.

1. If the content is a UI Element render it as is
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 data template to render it.
4. 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.

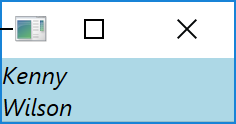
Figure 5Control and Data Templates



Listing 5Content and Data Templates



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.

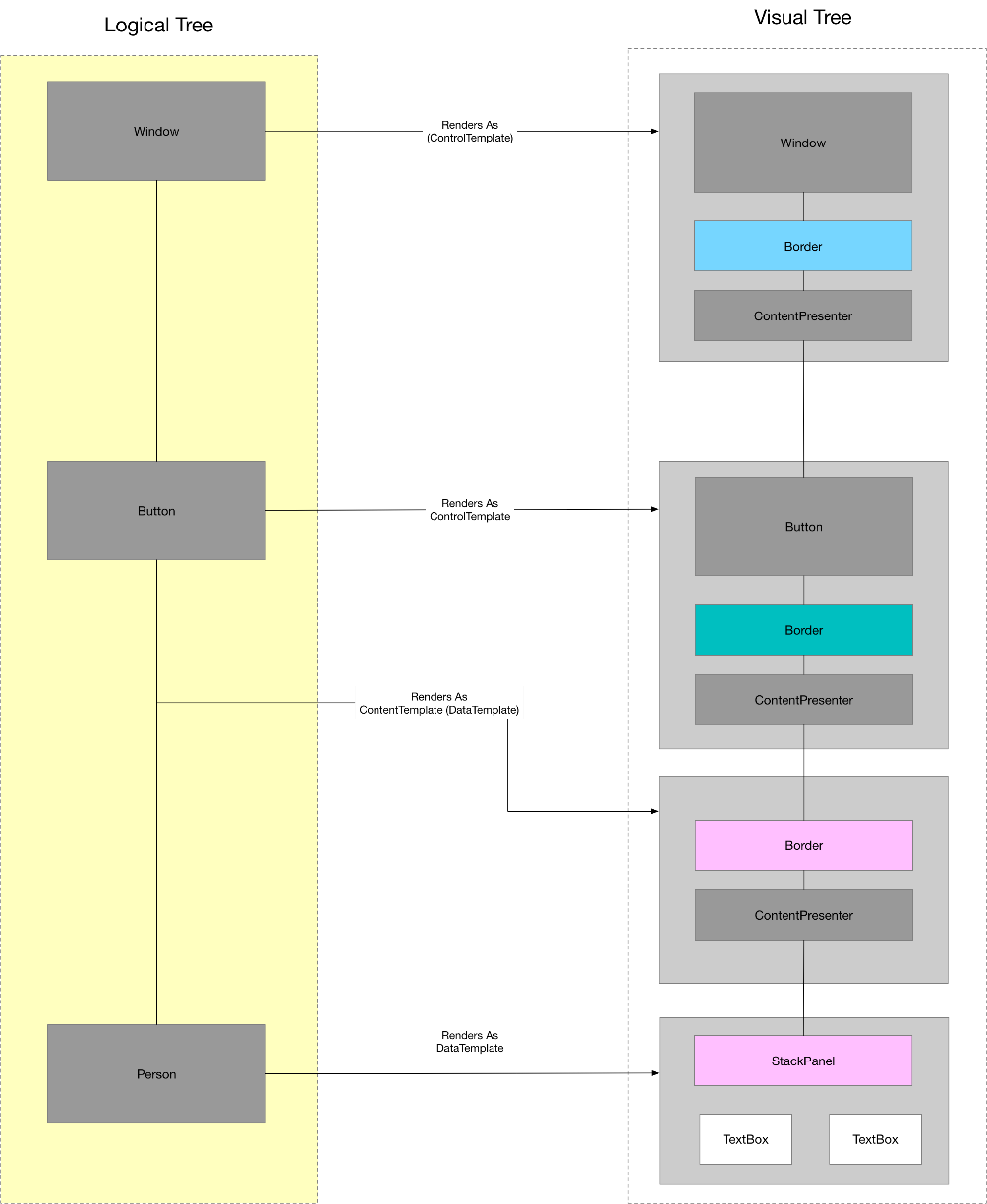
DataTemplates and ControlTemplates while similar (both extend the same base class) are intended for different purposes. A control template defines the look of the whole control and tends to be somewhat complex. A data template defines the look of a piece of content within a content control and tends 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



Figure 6ContentTemplate



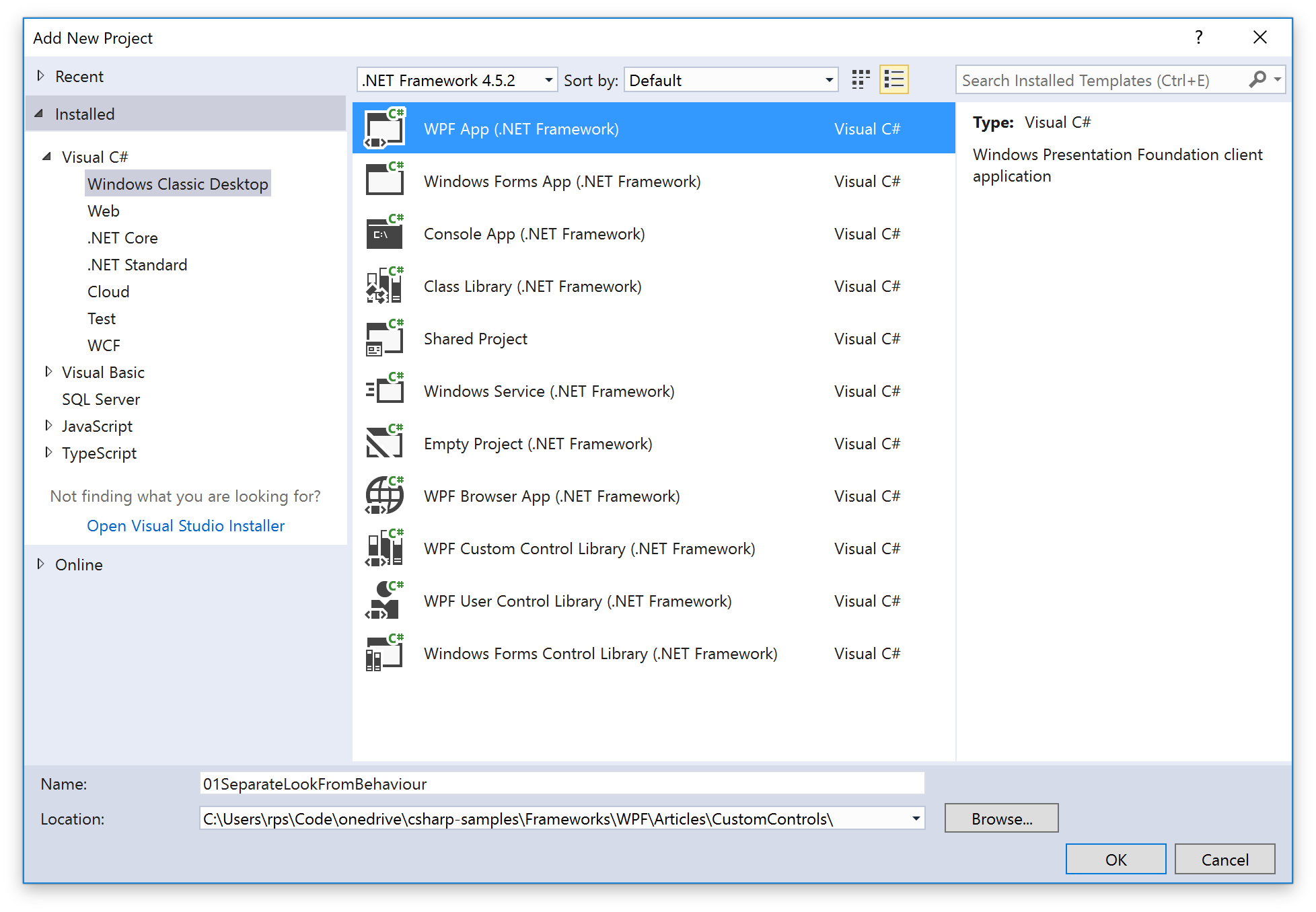
Appendices

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

We call our project 01SeparateLookFromBehaviour



1. Add a MyCustomControl class that extends Control



1. Add a static constructor with the following boilerplate



1. 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



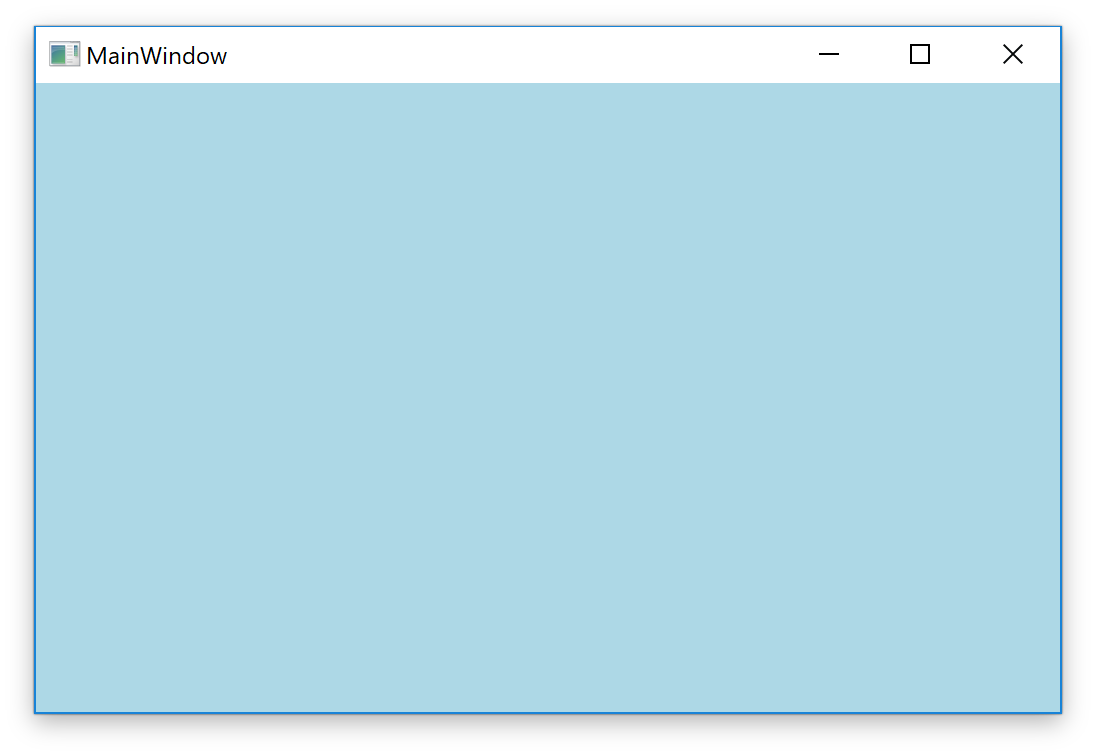
Because we have not created any template to defne the look our control is essentially lookless.

1. 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



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



1. Add the required part to the template



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



1. Create a simple view model



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



Questions

What defined a control?

Its behaviour

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

Setting properties directly such as foreground

With styles

With templates

Custom controls we can place graphics

Is a WPF control directly responsible for its visuals?

No it delegates to a template to perform its visuals

What is the benefit of this approach?

You need not write a custom control to change the appearance of a control

when do you need to write a custom control?

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

Are all UI elements in WPF controls?

No

What is the base class fort the UI elements that don’t have behviour?

FrameworkElement

Custom Controls