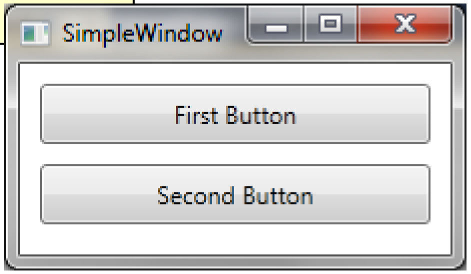
Control Templates

Logical Trees and Visual Trees

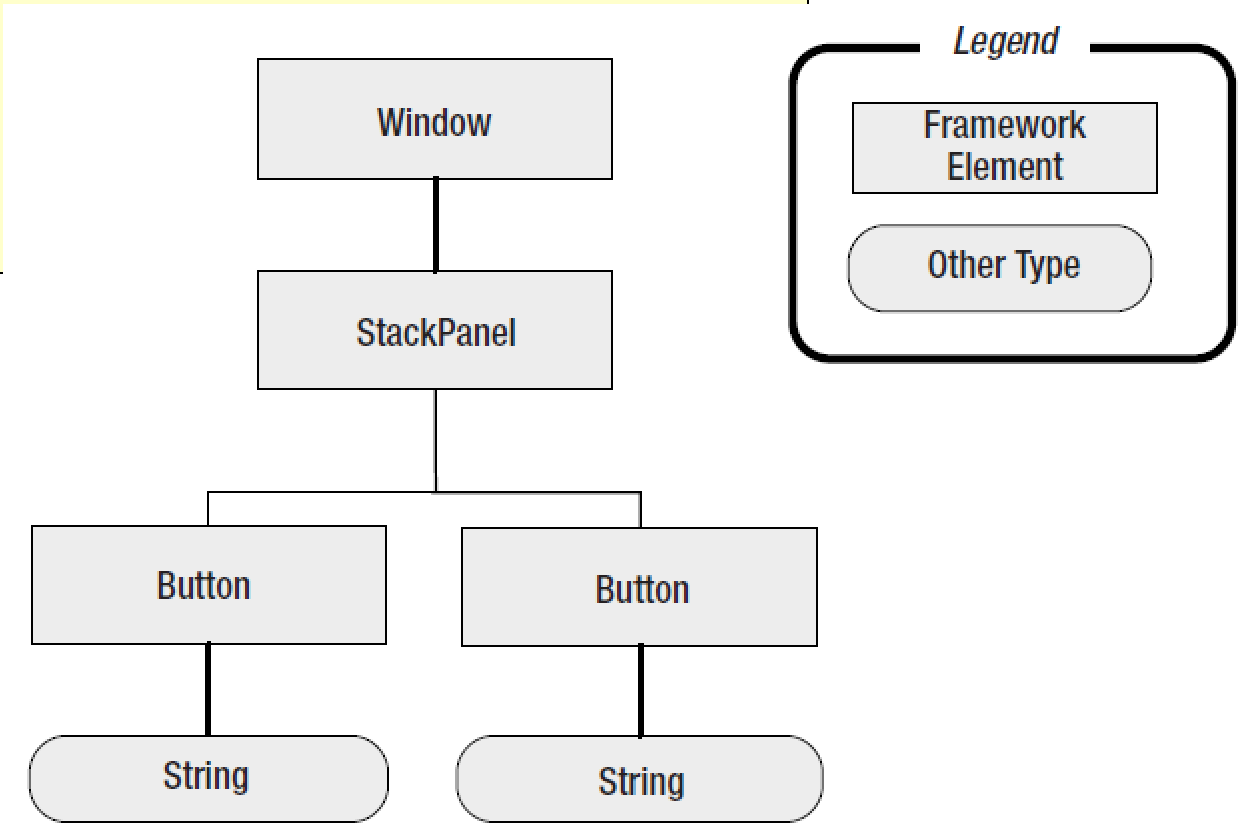
- A simple window with two buttons.

```
<Window x:Class="SimpleWindow.Window1"
        Title="SimpleWindow"
        >
  <StackPanel Margin="5">
    <Button Padding="5" Margin="5">First Button
    </Button>
    <Button Padding="5" Margin="5">Second Button
    </Button>
  </StackPanel>
</Window>
```



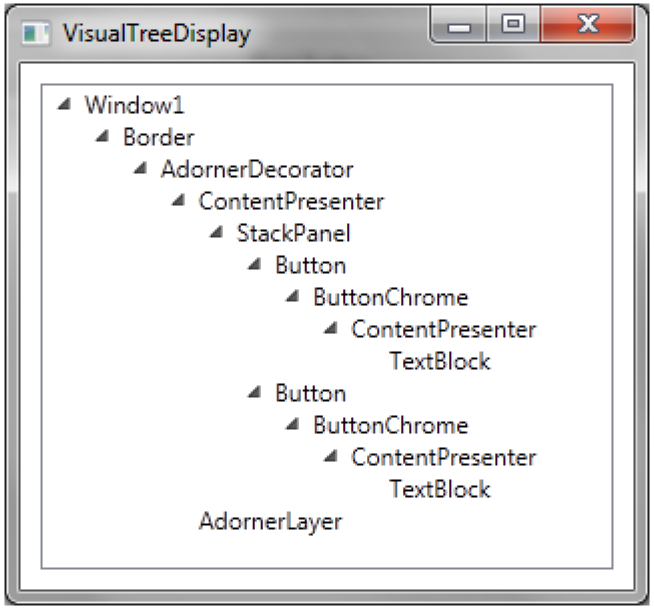
The Logical Tree

```
<Window x:Class="SimpleWindow.Window1"
        Title="SimpleWindow"
        >
  <StackPanel Margin="5">
    <Button Padding="5" Margin="5">First Button
    </Button>
    <Button Padding="
    </Button>
  </StackPanel>
</Window>
```

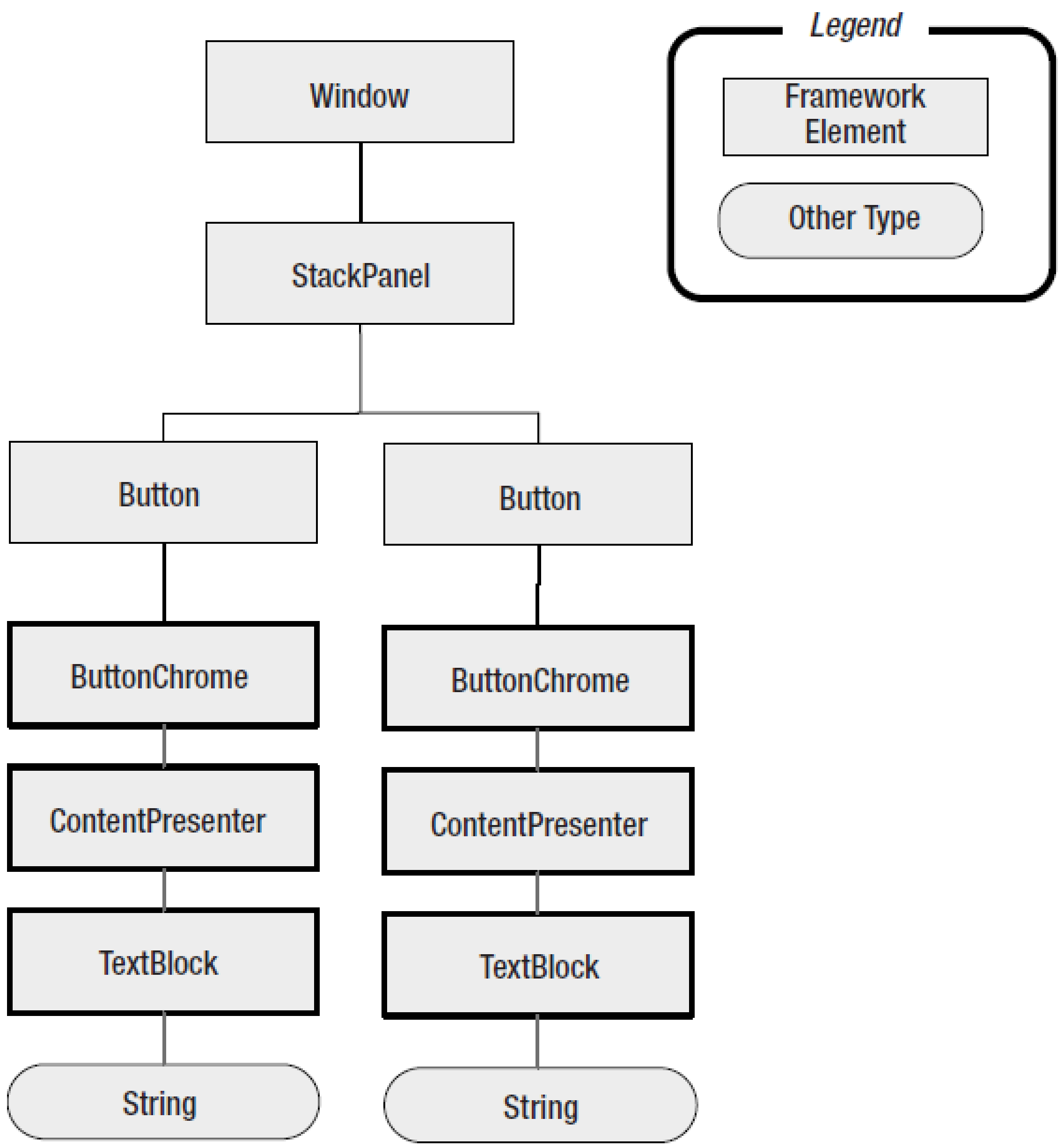


The Visual Tree

```
<Window x:Class="SimpleWindov
        Title="SimpleWindow"
        >
  <StackPanel Margin="5">
    <Button Padding="5" Margi
    </Button>
    <Button Padding="5" Margi
    </Button>
  </StackPanel>
</Window>
```

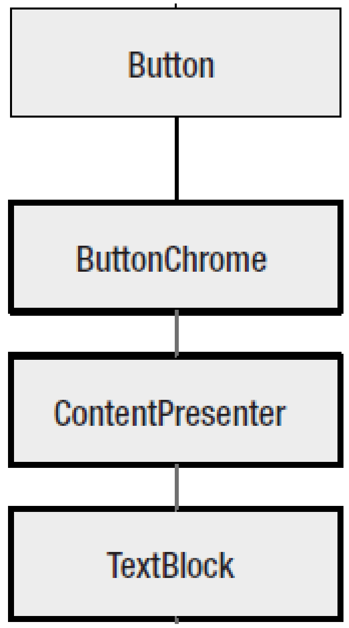


```
VisualTreeDisplay
◢ Window1
  ◢ Border
    ◢ AdornerDecorator
      ◢ ContentPresenter
        ◢ StackPanel
          ◢ Button
            ◢ ButtonChrome
              ◢ ContentPresenter
                  TextBlock
          ◢ Button
            ◢ ButtonChrome
              ◢ ContentPresenter
                  TextBlock
        AdornerLayer
```

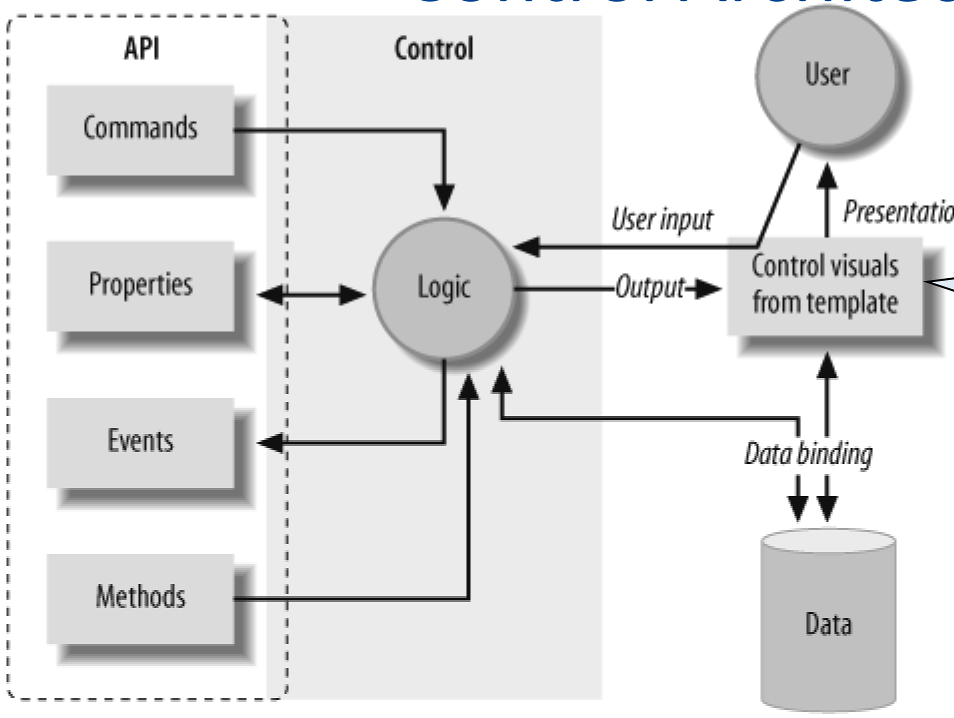


Understanding Templates

- How is a control translated from the logical tree into the expanded representation of the visual tree?
- Every control has a built-in recipe that determines how it should be rendered
- That recipe is called a ***control template***
- And it’s defined using a block of XAML markup.



Control Architecture



The control template is read from the Windows theme (e.g. Aero) unless you set it explicit in code.

- Controls in WPF uses a variation of the famous **M**odel **V**iew **C**ontroller design pattern.
  - The Model is attached to the Control by use of data binding
    - Model is a name for objects representing the underlying data.
  - **The template can be seen as the View**
    - **Contains the objects that display that data.**
  - And the control plays the role of the Controller
    - Contains objects that manage input from the user and interactions between the model and the view.

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

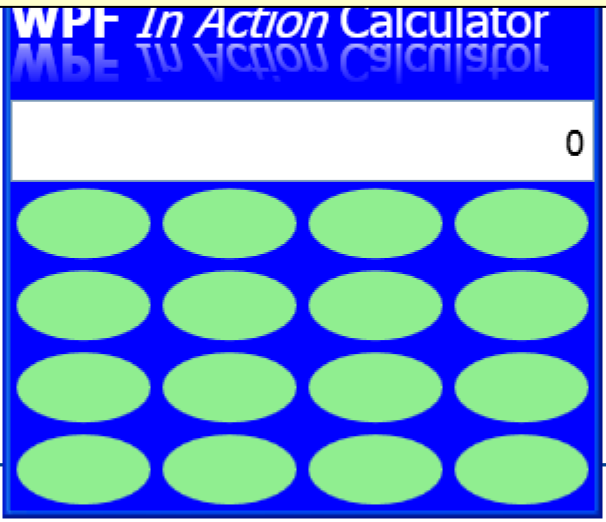
Control Templates

- **WPF** provides the ability to replace the complete look of the built-in controls while maintaining the existing behavior.
- The default look comes from the system-provided template.
- But we can override the default look by using our own "homemade" control template
  - a set of triggers, resources, and most important, elements that provide the look of a control.

A Style That Makes Buttons Ellipse

- One of the properties of a control is its template
  - Because the template is a property, it can be set as part of a style.

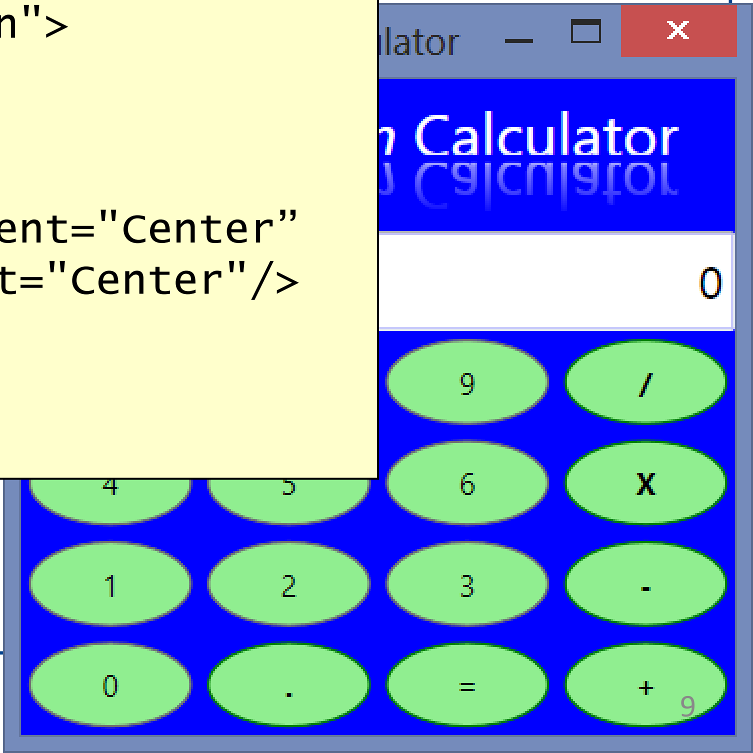
```
<Style x:Key="CalcButton" TargetType="Button">
   <Setter Property="Template">
      <Setter.Value>
         <ControlTemplate TargetType="Button">
            <Ellipse Fill="LightGreen" />
         </ControlTemplate>
      </Setter.Value>
   </Setter>
   <Setter Property="Control.Margin" Value="10"/>
</Style>
```



ContentPresenter

- The previous example has a slight problem
  - The text on the buttons (their content) is missing.
- To make the text show up again
  - we need to tell the system where to put that content.
- When you put a **ContentPresenter** into a control template
  - WPF shoves the control’s content wherever the content presenter says.

```
<Style x:Key="CalcButton" TargetType="Button">
   <ControlTemplate TargetType="Button">
      <Grid>
         <Ellipse Fill="LightGreen"/>
         <ContentPresenter HorizontalAlignment="Center”
                           VerticalAlignment="Center"/>
      </Grid>
   </ControlTemplate>
</Style>
```



Template Binding

- You can bind any of the properties to the value set on the control.
- If there’s code already setting a particular color on the button, you can use that color in the control template.
- Let’s use that to set the border around the ellipse:

```
<Style x:Key="CalcButton" TargetType="Button">
   <ControlTemplate TargetType="Button">
      <Grid>
        <Ellipse Fill="LightGreen”
                 Stroke="{TemplateBinding Control.BorderBrush}"/>
        <ContentPresenter HorizontalAlignment="Center”
                          VerticalAlignment="Center"/>
      </Grid>
   </ControlTemplate>
</Style>
```

Triggers

- We can make the buttons react to the user by adding triggers to the control template.

```
<Style x:Key="CalcButton" TargetType="Button">
  <Setter Property="Template">
    <Setter.Value>
       <ControlTemplate TargetType="Button">
         <Grid>
           <Ellipse x:Name="theEllipse" Fill="LightGreen”
                Stroke="{TemplateBinding Control.BorderBrush}"/>
           <ContentPresenter HorizontalAlignment="Center”
                              VerticalAlignment="Center"/>
         </Grid>
       <ControlTemplate.Triggers>
         <Trigger Property="Button.IsPressed" Value="True" >
           <Setter TargetName="theEllipse”
                   Property="Fill" Value="Yellow"/>
         </Trigger>
       </ControlTemplate.Triggers>
    </ControlTemplate>
</Setter.Value>
</Setter>
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

VISUAL STATES

Visual States

- Trigger-based templates have a downside:
  - they require that the template designer has a detailed understanding of the way the control works.
  - And it often gets very complicated for fancy controls with many states.
- Using named parts and visual states, a control can provide a standardized visual contract.
- Button provides this set of visual states:

```
[TemplateVisualState(Name="Normal", GroupName="CommonStates")]
[TemplateVisualState(Name="MouseOver", GroupName="CommonStates")]
[TemplateVisualState(Name="Pressed", GroupName="CommonStates")]
[TemplateVisualState(Name="Disabled", GroupName="CommonStates")]
[TemplateVisualState(Name="Unfocused", GroupName="FocusStates")]
[TemplateVisualState(Name="Focused", GroupName="FocusStates")]
```

Links

- Snoop Tool
  http://snoopwpf.codeplex.com/
- WPF Control Templates - An Overview
  http://blogs.msdn.com/b/jitghosh/archive/2007/12/27/wpf-control-templates-an-overview.aspx
- ControlTemplate Class
  http://msdn.microsoft.com/en-us/library/system.windows.controls.controltemplate(v=vs.110).aspx
- The VisualStateManager and Triggers
  http://blogs.msdn.com/b/wpfsdk/archive/2009/02/27/the-visualstatemanager-and-triggers.aspx