Шаблоны элементов управления

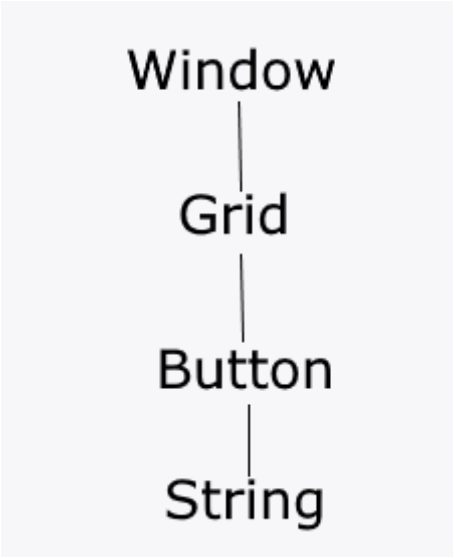
Логическое и визуальное дерево

Несмотря на то, что стили существенно облегчают манипулирование внешним видом элементов управления, гораздо более сильным средством в плане визуализации являются шаблоны. В отличие от стилей шаблоны помогают полностью менять модель визуализации элемента. Чтобы понять шаблоны, необходимо осознать общую концепцию визуализации в WPF.

Визуализация в WPF тесно связана с такими понятиями как логическое и визуальное дерево. Эти деревья являются своего рода каркасом приложения. Так мы можем представить приложение как некий набор вложенных элементов. Возьмем, к примеру, следующую простейшую разметку xaml:

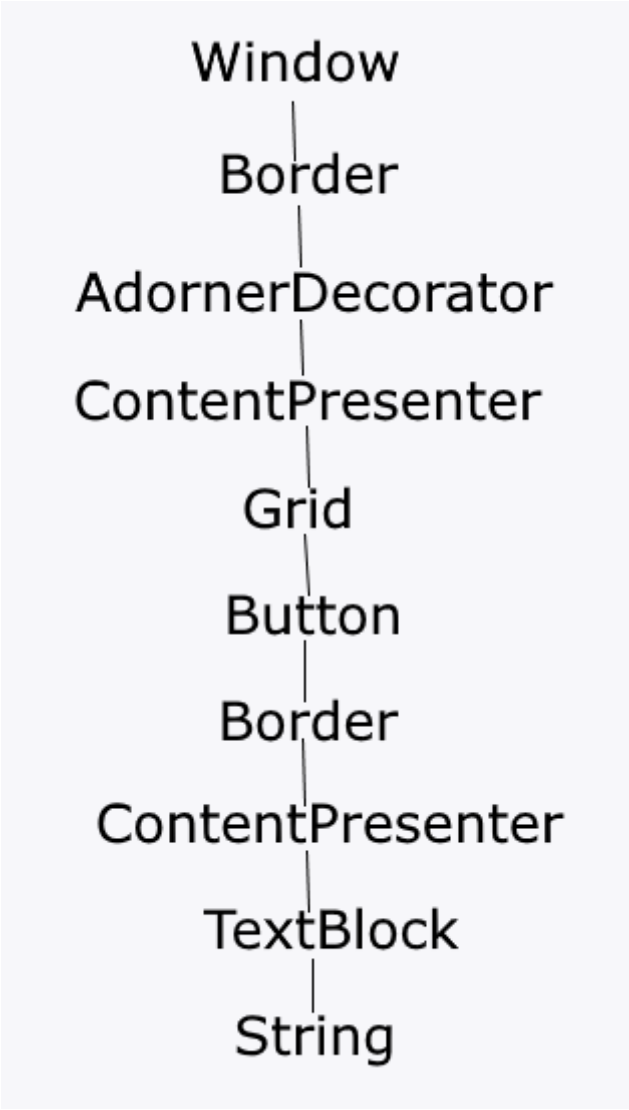
```
<Window x:Class="ControlsTemplateApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:ControlsTemplateApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Grid>
        <Button x:Name="myButton">Hello</Button>
    </Grid>
</Window>
```

Структуру элементов здесь можно представить следующей схемой:



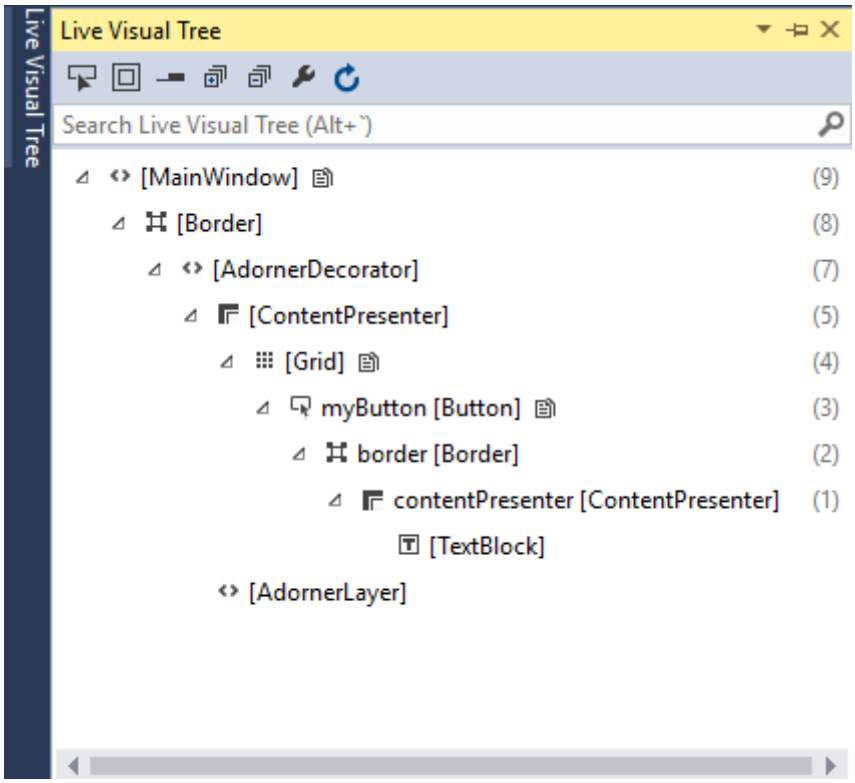
То есть в Window есть Grid, в Gride - элемент Button, в кнопке в качестве содержимого установлен некоторый текст в виде объекта String. В итоге получается некое дерево элементов, которое называется **логическим**. В WPF оно представлено классом **System.Windows.LogicalTreeHelper**. Логическое дерево имеет дело с визуализацией как таковой, оно образует модель доступа к дочерним элементам.

От него отличается **визуальное дерево**, представленное классом **System.Windows.Media.VisualTreeHelper**. Так, визуальное дерево для вышеприведенной разметки xaml будет выглядеть следующим образом:

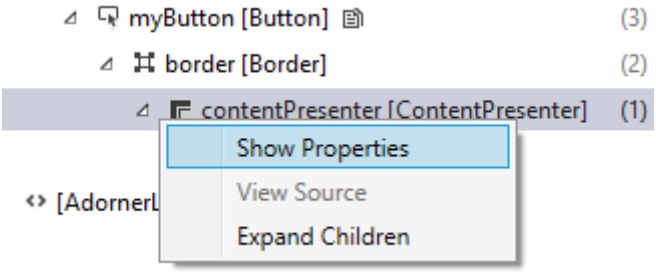


Визуальное дерево получается гораздо сложнее, оно показывает, как с визуальной точки зрения устроен элемент, из каких частей он состоит.

Visual Studio имеет встроенные средства для просмотра визуального дерева элементов. Для этого нам надо запустить проект в режиме отладки и в меню выбрать пункт **Debug -> Windows -> Live Visual Tree**. После нажатия на этот пункт Visual Studio откроет окно с визуальным деревом, в котором мы можем посмотреть, как устроен элемент:



Если мы нажмем правой кнопкой мыши на элемент в этом дереве и в контекстном меню выберем пункт **Show Properties**, справа в Visual Studio откроется окно свойств для этого элемента.



Визуальное дерево элемента управления опредлеляет, как будет выглядеть этот элемент или иными словами его **шаблон**. Шаблон элемента - это своего рода визуальный скелет элемента управления. Например, для элемента Button упрощенно шаблон выглядит следующим образом:

```
<Style TargetType="Button">
  <Setter Property="SnapsToDevicePixels" Value="true"/>
  <Setter Property="OverridesDefaultStyle" Value="true"/>
  <Setter Property="FocusVisualStyle" Value="{StaticResource ButtonFocusVisual}"/>
  <Setter Property="MinHeight" Value="23"/>
  <Setter Property="MinWidth" Value="75"/>
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="Button">
        <Border
          x:Name="Border"
          CornerRadius="2"
          BorderThickness="1"
          Background="{StaticResource NormalBrush}"
          BorderBrush="{StaticResource NormalBorderBrush}">
          <ContentPresenter
            Margin="2"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            RecognizesAccessKey="True"/>
        </Border>
        <ControlTemplate.Triggers>
          <Trigger Property="IsKeyboardFocused" Value="true">
            <Setter TargetName="Border" Property="BorderBrush" Value="{StaticResource DefaultedB
          </Trigger>
          <Trigger Property="IsDefaulted" Value="true">
            <Setter TargetName="Border" Property="BorderBrush" Value="{StaticResource DefaultedB
          </Trigger>
          <Trigger Property="IsMouseOver" Value="true">
            <Setter TargetName="Border" Property="Background" Value="{StaticResource DarkBrush}"
          </Trigger>
          <Trigger Property="IsPressed" Value="true">
            <Setter TargetName="Border" Property="Background" Value="{StaticResource PressedBrus
            <Setter TargetName="Border" Property="BorderBrush" Value="{StaticResource PressedBor
          </Trigger>
          <Trigger Property="IsEnabled" Value="false">
            <Setter TargetName="Border" Property="Background" Value="{StaticResource DisabledBac
            <Setter TargetName="Border" Property="BorderBrush" Value="{StaticResource DisabledBc
            <Setter Property="Foreground" Value="{StaticResource DisabledForegroundBrush}"/>
          </Trigger>
        </ControlTemplate.Triggers>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Полное описание шаблона кнопки можно найти по ссылке https://msdn.microsoft.com/en-us/library/ms753328(v=vs.100).aspx.

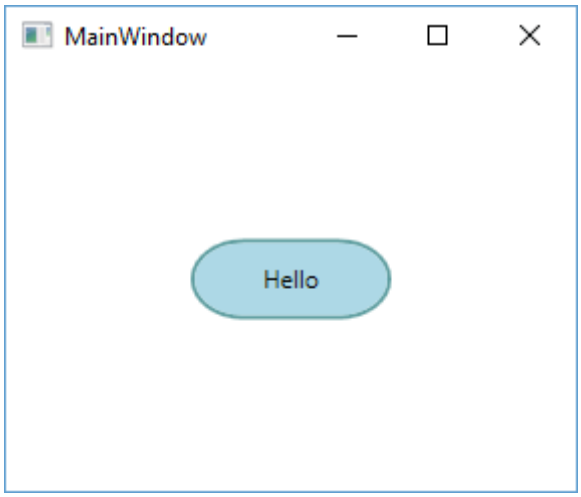
Здесь мы можем увидеть, что сам шаблон задается с помощью стиля при определении сеттера для свойства **Template**. Чтобы определить сам, шаблон, применяется элемент **ControlTemplate**. И как раз в этом элементе мы можем увидеть и элемент Border, и элемент ContentPresenter, которые составляют визуальное дерево кнопки.

Создание и использование шаблонов

Все визуальные элементы в WPF уже имеют встроенные шаблоны, которые определяют визуальное дерево, структуру и даже поведение элементов. Однако мощь шаблонов состоит в том, что мы можем их переопределить по своему вкусу. Например, сделать круглое окно, а не квадратное, или кнопку в виде морской звезды.

К примеру создадим округлый элемент Button:

```
<Window x:Class="ControlsTemplateApp.MainWindow"
        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:local="clr-namespace:ControlsTemplateApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Window.Resources>
        <ControlTemplate TargetType="Button" x:Key="btTemplate">
            <Border CornerRadius="25" BorderBrush="CadetBlue" BorderThickness="2"
                    Background="LightBlue" Height="40" Width="100" >
                <ContentControl Margin="5" HorizontalAlignment="Center" VerticalAlignment="Cente
            </Border>
        </ControlTemplate>
    </Window.Resources>
    <Grid>
        <Button x:Name="myButton" Template="{StaticResource btTemplate}" Height="40" Width="100"
    </Grid>
</Window>
```



Мы можем определять шаблоны с через стили, а можем в виде отдельных ресурсов. Так, в данном случае с помощью элемента **ControlTemplate** определяется ресурс с ключом "btTemplate".

В ControlTemplate вложены элементы Border и ContentControl, которые через свои свойства определяют, как будет выглядеть кнопка.

TemplateBinding

Вышеопределенный шаблон устанавливал цвет, ряд других параметров, которые мы не могли бы изменить. Например, если мы установим в кнопке цвет фона, то этот цвет никак не повлияет, так как в реальности будет работать только цвет, определенный в элементе Border:

```
<Button x:Name="myButton" Template="{StaticResource btTemplate}" Background="Red">Привет</Buttor
```

Чтобы мы могли из элемента, к которому применяется шаблон, влиять на свойства, определенные в этом шаблоне, нам надо использовать элемент **TemplateBinding**. Он служит для установки в шаблоне привязки к свойствам элемента. Так, изменим шаблон следующим образом:

```
<Window x:Class="ControlsTemplateApp.MainWindow"
        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:local="clr-namespace:ControlsTemplateApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Window.Resources>
        <ControlTemplate TargetType="Button" x:Key="btTemplate">
            <Border CornerRadius="25"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    BorderThickness="{TemplateBinding BorderThickness}"
                    Background="{TemplateBinding Background}"
                    Height="{TemplateBinding Height}"
                    Width="{TemplateBinding Width}" >
                <ContentControl Margin="{TemplateBinding Padding}"
                                HorizontalAlignment="{TemplateBinding HorizontalContentAlignment
                                VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                                Content="{TemplateBinding Content}" />
            </Border>
        </ControlTemplate>
    </Window.Resources>
    <Grid>
        <Button x:Name="myButton" Template="{StaticResource btTemplate}" Height="40" Width="100"
    </Grid>
</Window>
```

Выражение типа Background="{TemplateBinding Background}" в элементе Border указывает, что фон элемента Border будет привязан к свойству Background элемента Button. Таким образом, здесь практически все свойства Border и ContentControl (кроме свойства CornerRadius) устанавливаются из разметки элемента Button

Свойство Template

Используя свойство **Template**, можно определить шаблон напрямую в самом элементе:

```
<Button x:Name="myButton" Content="Привет" Height="40" Width="100" Background="LightPink">
    <Button.Template>
        <ControlTemplate TargetType="Button">
            <Border CornerRadius="25"
                BorderBrush="{TemplateBinding BorderBrush}"
                BorderThickness="{TemplateBinding BorderThickness}"
                Background="{TemplateBinding Background}"
                Height="{TemplateBinding Height}"
                Width="{TemplateBinding Width}" >
                <ContentControl Margin="{TemplateBinding Padding}"
                        HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                        VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                        Content="{TemplateBinding Content}" />
            </Border>
        </ControlTemplate>
    </Button.Template>
</Button>
```

Триггеры в шаблонах

Если мы возьмем стандартные кнопки (или другие элементы управления), то мы можем заметить, что они меняют свои свойства в различных состояниях. Например, кнопка изменяет фон при нажатии. Рассмотрим, как мы сможем сделать подобные вещи.

Для реагирования на изменение свойств в шаблонах, как и в стилях, используютс триггеры. Для триггеров в элементе ControlTemplate определена коллекция **Triggers**. Например, добавим к ранее определенному шаблону несколько триггеров:

```
<Button x:Name="myButton" Content="Привет" Height="40" Width="100" Background="LightPink">
    <Button.Template>
        <ControlTemplate TargetType="Button">
            <Border x:Name="buttonBorder" CornerRadius="25"
                BorderBrush="{TemplateBinding BorderBrush}"
                BorderThickness="{TemplateBinding BorderThickness}"
                Background="{TemplateBinding Background}"
                Height="{TemplateBinding Height}"
                Width="{TemplateBinding Width}" >
                <ContentControl Margin="{TemplateBinding Padding}"
                    HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                    VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                    Content="{TemplateBinding Content}" />
            </Border>
            <ControlTemplate.Triggers>
                <Trigger Property="IsMouseOver" Value="true">
                    <Setter Property="FontWeight" Value="Bold" />
                </Trigger>
                <Trigger Property="IsPressed" Value="true">
                    <Setter TargetName="buttonBorder" Property="Background" Value="Azure" />
                    <Setter TargetName="buttonBorder" Property="BorderBrush" Value="DarkBlue" />
                </Trigger>
                <Trigger Property="IsEnabled" Value="false">
                    <Setter Property="Foreground" Value="Gray"/>
                    <Setter TargetName="buttonBorder" Property="Background" Value="LightGray"/>
                </Trigger>
            </ControlTemplate.Triggers>
        </ControlTemplate>
    </Button.Template>
</Button>
```

Здесь для элемента Border определено свойство x:Name="buttonBorder", благодаря чему мы можем ссылаться на этот элемент в триггерах.

В коллекции триггеров шаблона определено три триггера. Первый срабатывает, когда свойство **IsMouseOver** элемена Button будет равно true. То есть при наведении указателя мыши на кнопку. В этом случае триггер делает жирным шрифт кнопки:

```
<Trigger Property="IsMouseOver" Value="true">
    <Setter Property="FontWeight" Value="Bold" />
</Trigger>
```

Второй триггер срабатывает при нажатии на кнопку, а третий - когда свойство **IsEnabled** равно false. Причем в этих триггерах свойство **TargetName**, что позволяет установить свойства не самой кнопки, а элемента Border в шаблоне.

Визуальные состояния

Элемент управления обладает определенными состояниями, которые могут влиять на визуальное отображение. Например, для кнопки это может быть состояние **MouseOver**, когда указатель мыши наведен на кнопку, или состояние **Pressed**, когда кнопка нажата. У большинства элементов управления есть несколько состояний. И мы можем управлять переходом в эти состояния.

Чтобы изменить внешний вид в зависимости от состояния элемента используется объект **ViewState**. Объект ViewState через другой объект - **Storyboard** позволяет определить анимацию, изменяющую внешний вид элемента.

Так, используем несколько состояний, определенных в элементе Button:

```
<Button x:Name="myButton" Content="Привет" Height="40" Width="100">
    <Button.Template>
        <ControlTemplate TargetType="Button">
            <Border CornerRadius="25"
                BorderBrush="{TemplateBinding BorderBrush}"
                BorderThickness="{TemplateBinding BorderThickness}"
                Height="{TemplateBinding Height}"
                Width="{TemplateBinding Width}">

                <Border.Background>
                    <SolidColorBrush x:Name="BorderColor" Color="LightPink" />
                </Border.Background>
                <ContentControl Margin="{TemplateBinding Padding}"
                    HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                    VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                    Content="{TemplateBinding Content}" />

                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup Name="CommonStates">
                        <VisualState Name="MouseOver">
                            <Storyboard>
                                <ColorAnimation Storyboard.TargetName="BorderColor"
                                    Storyboard.TargetProperty="Color" To="LightBlue" />
                            </Storyboard>
                        </VisualState>
                        <VisualState Name="Normal">
                            <Storyboard>
                                <ColorAnimation Storyboard.TargetName="BorderColor"
                                    Storyboard.TargetProperty="Color" To="LightPink" />
                            </Storyboard>
                        </VisualState>
                    </VisualStateGroup>
                </VisualStateManager.VisualStateGroups>
            </Border>
        </ControlTemplate>
    </Button.Template>
</Button>
```

При применении данного шаблона при наведении мыши на кнопку она будет приобретать светло-синий цвет, а если мышь покинет пределы кнопки, цвет опять станет розовой.

За счет чего это происходит? Корневой элемент Border имеет объект **VisualStateManager**, у которого, в свою очередь, есть коллекция объектов **VisualStateGroup**. В данном случае у нас используется только один объект VisualStateGroup - **CommonStates**. Данная группа содержит такие состояния как **MouseOver**, **Pressed**, **Normal**. А объекты VisualState как раз и задают состояния и применяемую к ним анимацию.

Анимацией управляет объект **Storyboard**. Его свойство **TargetName** указывает на имя объекта, который анимируется, а **TargetProperty** - свойство объекта, которое анимируется.

Если объект VisualState используется для визуализации элемента в определенном состоянии, то объект **VisualTransition** визуализирует элемент в момент перехода от одного состояния в другое. Так, если мы задали два визуальных состояния, то мы можем определить визуальный переход из одного состояния в другое:

```
<Button x:Name="myButton" Content="Привет" Height="40" Width="100">
    <Button.Template>
        <ControlTemplate TargetType="Button">
            <Border CornerRadius="25"
                BorderBrush="{TemplateBinding BorderBrush}"
                BorderThickness="{TemplateBinding BorderThickness}"
                Height="{TemplateBinding Height}"
                Width="{TemplateBinding Width}">

                <Border.Background>
                    <SolidColorBrush x:Name="BorderColor" Color="LightPink" />
                </Border.Background>
                <ContentControl Margin="{TemplateBinding Padding}"
                    HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                    VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                    Content="{TemplateBinding Content}" />

                <VisualStateManager.VisualStateGroups>
                    <VisualStateGroup Name="CommonStates">
                        <VisualState Name="MouseOver">
                            <Storyboard>
                                <ColorAnimation Storyboard.TargetName="BorderColor"
                                    Storyboard.TargetProperty="Color" To="LightBlue" />
                            </Storyboard>
                        </VisualState>
                        <VisualState Name="Normal">
                            <Storyboard>
                                <ColorAnimation Storyboard.TargetName="BorderColor"
                                    Storyboard.TargetProperty="Color" To="LightPink" />
                            </Storyboard>
                        </VisualState>
                        <VisualStateGroup.Transitions>
                            <VisualTransition From="MouseOver" To="Normal"
                                    GeneratedDuration="0:0:1.5">
                                <Storyboard>
                                    <ColorAnimationUsingKeyFrames Storyboard.TargetName="BorderC
    Storyboard.TargetProperty="Color" FillBehavior="HoldEnd">
  <ColorAnimationUsingKeyFrames.KeyFrames>
    <LinearColorKeyFrame Value="Yellow" KeyTime="0:0:0.5" />
    <LinearColorKeyFrame Value="Red" KeyTime="0:0:1" />
    <LinearColorKeyFrame Value="Green" KeyTime="0:0:1.5" />
  </ColorAnimationUsingKeyFrames.KeyFrames>
                                </ColorAnimationUsingKeyFrames>
                            </Storyboard>
                        </VisualTransition>
                    </VisualStateGroup.Transitions>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </Border>
    </ControlTemplate>
```

```
    </Button.Template>
</Button>
```

В данном случае в момент перехода опять же анимируется фон кнопки, только теперь вся анимация длится 1.5 секунды, в течение которых кнопка последовательно приоретает желтый, красный и зеленый цвет, после чего возвращается в исходное состояние.

Круглое окно

Для практики с шаблонами переопределим шаблон окна, чтобы оно имело нестандартную непрямоугольную форму:

```
<Window x:Class="ControlsTemplateApp.MainWindow"
        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:local="clr-namespace:ControlsTemplateApp"
        mc:Ignorable="d"
        Title="Круглое окно" Height="250" Width="300"
        AllowsTransparency="True" WindowStyle="None" Background="Transparent">
    <Window.Template>
        <ControlTemplate TargetType="Window">
        <Border Name="newBorder" CornerRadius="150" Opacity="0.7" Background="LightBlue">
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>
                <!--Заголовок-->
                <TextBlock Text="{TemplateBinding Title}" FontWeight="Bold" HorizontalAlignment=
                    MouseLeftButtonDown="TextBlock_MouseLeftButtonDown"/>
                <!--Основное содержание-->
                    <Border Grid.Row="1">
                        <AdornerDecorator>
                            <ContentPresenter />
                        </AdornerDecorator>
                    </Border>
                    <!--Элемент захвата и изменения размера - работает только для прямоугольных
                <ResizeGrip Grid.Row="2" HorizontalAlignment="Right" VerticalAlignment="Bottom"
                            Visibility="Collapsed" IsTabStop="False" />
            </Grid>
        </Border>
    </ControlTemplate>
    </Window.Template>
    <Grid>
        <Button x:Name="closeButton" Content="Закрыть" Click="closeButton_Click"
                Width="80" Height="30" Background="LightPink" />
    </Grid>
</Window>
```

Важно здесь отметить установку трех свойств элемента Window:

- AllowsTransparency: позволяет сделать форму прозрачной, невидимой
- WindowStyle: позволяет убрать у окна стиль при установке значения "None"
- Background: устанавливает прозрачнй фон с помощью значения Transparent

И также в файле кода c# изменим код окна, определив необходимые обработчики событий:

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void closeButton_Click(object sender, RoutedEventArgs e)
    {
        this.Close();
    }

    //Служит для перемещения окна за заголовок формы
    private void TextBlock_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
    {
        this.DragMove();
    }
}
```