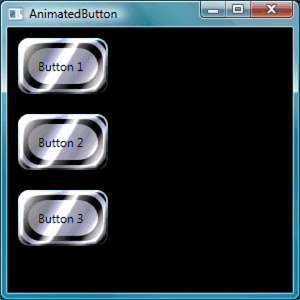
# Создание кнопки с помощью XAML

Цель этого пошагового руководства — научиться создавать анимированную кнопку для использования в приложении Windows Presentation Foundation (WPF). В этом пошаговом руководстве используются стили и шаблон для создания настраиваемого ресурса кнопки, который позволяет повторно использовать код и разделение логики кнопки из объявления кнопки. Это пошаговое руководство написано полностью в XAML (XAML).

На следующем рисунке показаны готовые кнопки.



## Создание основных кнопок

Начнем с создания нового проекта и добавления нескольких кнопок в окно.

### Создание нового проекта WPF и добавление в него кнопок

1. Запустите среду Visual Studio.
2. **Создайте новый проект WPF:** В меню **файл** наведите указатель мыши на пункт **создать**и выберите пункт **Project**. найдите шаблон **приложения Windows (WPF)** и присвойте проекту имя "аниматедбуттон". Это приведет к созданию каркаса для приложения.
3. **Добавить основные кнопки по умолчанию:** Все файлы, необходимые для этого пошагового руководства, предоставляются шаблоном. откройте файл Window1. xaml, дважды щелкнув его в Обозреватель решений. По умолчанию в Window1. XAML присутствует Grid элемент. Grid Удалите элемент и добавьте несколько кнопок на страницу XAML (XAML), введя или скопировав и вставляя следующий выделенный код в Window1. XAML:

XAML

<Window x:Class="AnimatedButton.Window1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="AnimatedButton" Height="300" Width="300"

Background="Black">

<!-- Buttons arranged vertically inside a StackPanel. -->

<StackPanel HorizontalAlignment="Left">

<Button>Button 1</Button>

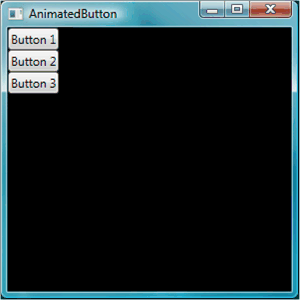
<Button>Button 2</Button>

<Button>Button 3</Button>

</StackPanel>

</Window>

Нажмите клавишу F5, чтобы запустить приложение. Вы увидите набор кнопок, которые выглядят как на следующем рисунке.



Теперь, когда основные кнопки созданы, вы завершили работу в файле Window1. XAML. Остальная часть пошагового руководства посвящена файлу app. XAML, определяющему стили и шаблон для кнопок.

## Задание основных свойств

Теперь давайте настроим некоторые свойства на этих кнопках, чтобы управлять внешним видом и макетом кнопки. Вместо того чтобы устанавливать свойства на кнопки по отдельности, вы будете использовать ресурсы для определения свойств кнопки для всего приложения. Ресурсы приложения концептуально похожи на внешние каскадные таблицы стилей (CSS) для веб-страниц. Тем не менее ресурсы являются гораздо более мощными, чем каскадные таблицы стилей (CSS), как вы увидите в конце этого пошагового руководства.

### Использование стилей для задания основных свойств кнопок

1. **Определите блок Application. Resources:** Откройте App. XAML и добавьте следующую выделенную разметку, если она еще не существует:

XAML

<Application x:Class="AnimatedButton.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

StartupUri="Window1.xaml"

>

<Application.Resources>

<!-- Resources for the entire application can be defined here. -->

</Application.Resources>

</Application>

Область ресурсов определяется местом, где определяется ресурс. Определение ресурсов в Application.Resources файле App. XAML позволяет использовать ресурс из любого места в приложении.

**Создайте стиль и определите в нем основные значения свойств:** Добавьте в блок следующую разметку Application.Resources . Эта разметка создает объект Style , который применяется ко всем кнопкам в приложении, присвоив параметрам Width для кнопок значение 90 и значение Margin 10.

XAMLКопировать

<Application.Resources>

<Style TargetType="Button">

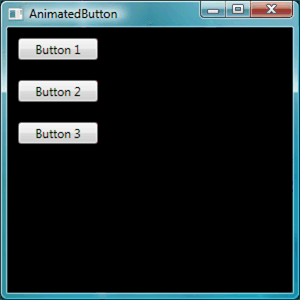
<Setter Property="Width" Value="90" />

<Setter Property="Margin" Value="10" />

</Style>

</Application.Resources>

TargetType Свойство указывает, что стиль применяется ко всем объектам типа Button . Каждый из Style них Setter задает другое значение свойства для. Таким образом, на этом этапе каждая кнопка в приложении имеет ширину 90 и поле, равное 10. Если нажать клавишу F5 для запуска приложения, появится следующее окно.



С помощью стилей можно выполнять множество других задач, в том числе различные способы точной настройки объектов, указания сложных значений свойств и даже использования стилей в качестве входных данных для других стилей.

1. **Задайте ресурсу значение свойства Style:** Ресурсы обеспечивают простой способ повторного использования часто заданных объектов и значений. Особенно полезно определить сложные значения с помощью ресурсов, чтобы сделать код более модульным. Добавьте следующую выделенную разметку в App. XAML.

XAMLКопировать

<Application.Resources>

<LinearGradientBrush x:Key="GrayBlueGradientBrush" StartPoint="0,0" EndPoint="1,1">

<GradientStop Color="DarkGray" Offset="0" />

<GradientStop Color="#CCCCFF" Offset="0.5" />

<GradientStop Color="DarkGray" Offset="1" />

</LinearGradientBrush>

<Style TargetType="{x:Type Button}">

<Setter Property="Background" Value="{StaticResource GrayBlueGradientBrush}" />

<Setter Property="Width" Value="80" />

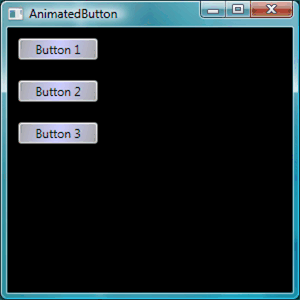
<Setter Property="Margin" Value="10" />

</Style>

</Application.Resources>

Непосредственно под Application.Resources блоком вы создали ресурс с именем «грайблуеградиентбруш». Этот ресурс определяет горизонтальный градиент. Этот ресурс можно использовать как значение свойства из любого места в приложении, включая внутри метода задания стиля кнопки для Background Свойства. Теперь все кнопки имеют Background значение свойства этого градиента.

Нажмите клавишу F5 для запуска приложения. Это должно выглядеть примерно так.



## Создание шаблона, определяющего вид кнопки

В этом разделе вы создадите шаблон, который настраивает внешний вид (представление) кнопки. Кнопка презентации состоит из нескольких объектов, включая прямоугольники и другие компоненты, чтобы дать кнопке уникальный вид.

До сих пор Управление тем, как выглядят кнопки в приложении, было ограничено изменением свойств кнопки. Что делать, если вы хотите внести более коренные изменения в внешний вид кнопки? Шаблоны обеспечивают мощный контроль над представлением объекта. Поскольку шаблоны можно использовать в стилях, можно применить шаблон ко всем объектам, к которым применяется стиль (в этом пошаговом руководстве это кнопка).

### Использование шаблона для определения вида кнопки

1. **Настройте шаблон:** Так как элементы управления, такие Button как, имеют Template свойство, можно определить значение свойства шаблона так же, как и другие значения свойств, заданные в Style с помощью Setter . Добавьте следующую выделенную разметку в стиль кнопки.

XAMLКопировать

<Application.Resources>

<LinearGradientBrush x:Key="GrayBlueGradientBrush"

StartPoint="0,0" EndPoint="1,1">

<GradientStop Color="DarkGray" Offset="0" />

<GradientStop Color="#CCCCFF" Offset="0.5" />

<GradientStop Color="DarkGray" Offset="1" />

</LinearGradientBrush>

<Style TargetType="{x:Type Button}">

<Setter Property="Background" Value="{StaticResource GrayBlueGradientBrush}" />

<Setter Property="Width" Value="80" />

<Setter Property="Margin" Value="10" />

<Setter Property="Template">

<Setter.Value>

<!-- The button template is defined here. -->

</Setter.Value>

</Setter>

</Style>

</Application.Resources>

1. **Изменить представление кнопки:** На этом этапе необходимо определить шаблон. Добавьте следующую выделенную разметку. Эта разметка определяет два Rectangle элемента с закругленными краями, за которыми следует DockPanel . DockPanelИспользуется для размещения ContentPresenter кнопки. ContentPresenter Отображает содержимое кнопки. В этом пошаговом руководстве содержимое является текстом ("Кнопка 1", "Кнопка 2", "Кнопка 3"). Все компоненты шаблона (прямоугольники и DockPanel ) размещаются внутри Grid .

XAMLКопировать

<Setter.Value>

<ControlTemplate TargetType="Button">

<Grid Width="{TemplateBinding Width}" Height="{TemplateBinding Height}" ClipToBounds="True">

<!-- Outer Rectangle with rounded corners. -->

<Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Stroke="{TemplateBinding Background}" RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent" />

<!-- Inner Rectangle with rounded corners. -->

<Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Stroke="Transparent" StrokeThickness="20" Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20" />

<!-- Present Content (text) of the button. -->

<DockPanel Name="myContentPresenterDockPanel">

<ContentPresenter x:Name="myContentPresenter" Margin="20" Content="{TemplateBinding Content}" TextBlock.Foreground="Black" />

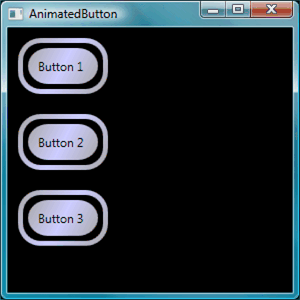
</DockPanel>

</Grid>

</ControlTemplate>

</Setter.Value>

Нажмите клавишу F5 для запуска приложения. Это должно выглядеть примерно так.



1. **Добавьте глассеффект в шаблон:** Далее вы добавите стекло. Сначала необходимо создать ресурсы, которые создают эффект прозрачного градиента. Добавьте эти ресурсы градиента в Application.Resources любом месте блока:

XAMLКопировать

<Application.Resources>

<GradientStopCollection x:Key="MyGlassGradientStopsResource">

<GradientStop Color="WhiteSmoke" Offset="0.2" />

<GradientStop Color="Transparent" Offset="0.4" />

<GradientStop Color="WhiteSmoke" Offset="0.5" />

<GradientStop Color="Transparent" Offset="0.75" />

<GradientStop Color="WhiteSmoke" Offset="0.9" />

<GradientStop Color="Transparent" Offset="1" />

</GradientStopCollection>

<LinearGradientBrush x:Key="MyGlassBrushResource"

StartPoint="0,0" EndPoint="1,1" Opacity="0.75"

GradientStops="{StaticResource MyGlassGradientStopsResource}" />

<!-- Styles and other resources below here. -->

Эти ресурсы используются в качестве Fill для прямоугольника, который мы вставляем в Grid шаблон кнопки. Добавьте в шаблон следующую выделенную разметку.

XAML

<Setter.Value>

<ControlTemplate TargetType="{x:Type Button}">

<Grid Width="{TemplateBinding Width}" Height="{TemplateBinding Height}"

ClipToBounds="True">

<!-- Outer Rectangle with rounded corners. -->

<Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch"

VerticalAlignment="Stretch" Stroke="{TemplateBinding Background}"

RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent" />

<!-- Inner Rectangle with rounded corners. -->

<Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch"

VerticalAlignment="Stretch" Stroke="Transparent" StrokeThickness="20"

Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20" />

<!-- Glass Rectangle -->

<Rectangle x:Name="glassCube" HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"

StrokeThickness="2" RadiusX="10" RadiusY="10" Opacity="0"

Fill="{StaticResource MyGlassBrushResource}"

RenderTransformOrigin="0.5,0.5">

<Rectangle.Stroke>

<LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">

<LinearGradientBrush.GradientStops>

<GradientStop Offset="0.0" Color="LightBlue" />

<GradientStop Offset="1.0" Color="Gray" />

</LinearGradientBrush.GradientStops>

</LinearGradientBrush>

</Rectangle.Stroke>

<!-- These transforms have no effect as they are declared here.

The reason the transforms are included is to be targets

for animation (see later). -->

<Rectangle.RenderTransform>

<TransformGroup>

<ScaleTransform />

<RotateTransform />

</TransformGroup>

</Rectangle.RenderTransform>

<!-- A BevelBitmapEffect is applied to give the button a "Beveled" look. -->

<Rectangle.BitmapEffect>

<BevelBitmapEffect />

</Rectangle.BitmapEffect>

</Rectangle>

<!-- Present Text of the button. -->

<DockPanel Name="myContentPresenterDockPanel">

<ContentPresenter x:Name="myContentPresenter" Margin="20"

Content="{TemplateBinding Content}" TextBlock.Foreground="Black" />

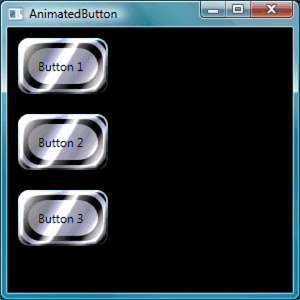
</DockPanel>

</Grid>

</ControlTemplate>

</Setter.Value>

Обратите внимание, что Opacity прямоугольник со x:Name свойством «гласскубе» имеет значение 0, поэтому при запуске образца прямоугольник с эффектом стекла не отображается сверху. Это связано с тем, что позже мы добавим триггеры к шаблону, когда пользователь взаимодействует с кнопкой. Тем не менее можно увидеть, как выглядит кнопка сейчас, изменив Opacity значение на 1 и запустив приложение. См. следующий рисунок. Прежде чем перейти к следующему шагу, измените Opacity обратно на 0.



## Взаимодействие кнопки "создать"

В этом разделе вы создадите триггеры свойств и триггеры событий для изменения значений свойств и запуска анимации в ответ на действия пользователя, такие как перемещение указателя мыши над кнопкой и нажатие кнопки.

Простой способ добавления интерактивности (нажатие кнопки мыши, выхода из мыши, щелчка и т. д.) заключается в определении триггеров в шаблоне или стиле. Чтобы создать объект Trigger , определите свойство "Condition", например: значение свойства кнопки IsMouseOver равно true . Затем определяются методы задания (Actions), которые выполняются, когда условие триггера имеет значение true.

### Создание интерактивной активности кнопки

1. **Добавить триггеры шаблона:** Добавьте выделенную разметку в шаблон.

XAMLКопировать

<Setter.Value>

<ControlTemplate TargetType="{x:Type Button}">

<Grid Width="{TemplateBinding Width}"

Height="{TemplateBinding Height}" ClipToBounds="True">

<!-- Outer Rectangle with rounded corners. -->

<Rectangle x:Name="outerRectangle" HorizontalAlignment="Stretch"

VerticalAlignment="Stretch" Stroke="{TemplateBinding Background}"

RadiusX="20" RadiusY="20" StrokeThickness="5" Fill="Transparent" />

<!-- Inner Rectangle with rounded corners. -->

<Rectangle x:Name="innerRectangle" HorizontalAlignment="Stretch"

VerticalAlignment="Stretch" Stroke="Transparent"

StrokeThickness="20"

Fill="{TemplateBinding Background}" RadiusX="20" RadiusY="20"

/>

<!-- Glass Rectangle -->

<Rectangle x:Name="glassCube" HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"

StrokeThickness="2" RadiusX="10" RadiusY="10" Opacity="0"

Fill="{StaticResource MyGlassBrushResource}"

RenderTransformOrigin="0.5,0.5">

<Rectangle.Stroke>

<LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">

<LinearGradientBrush.GradientStops>

<GradientStop Offset="0.0" Color="LightBlue" />

<GradientStop Offset="1.0" Color="Gray" />

</LinearGradientBrush.GradientStops>

</LinearGradientBrush>

</Rectangle.Stroke>

<!-- These transforms have no effect as they

are declared here.

The reason the transforms are included is to be targets

for animation (see later). -->

<Rectangle.RenderTransform>

<TransformGroup>

<ScaleTransform />

<RotateTransform />

</TransformGroup>

</Rectangle.RenderTransform>

<!-- A BevelBitmapEffect is applied to give the button a

"Beveled" look. -->

<Rectangle.BitmapEffect>

<BevelBitmapEffect />

</Rectangle.BitmapEffect>

</Rectangle>

<!-- Present Text of the button. -->

<DockPanel Name="myContentPresenterDockPanel">

<ContentPresenter x:Name="myContentPresenter" Margin="20"

Content="{TemplateBinding Content}" TextBlock.Foreground="Black" />

</DockPanel>

</Grid>

<ControlTemplate.Triggers> <!-- Set action triggers for the buttons and define what the button does in response to those triggers. --> </ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

1. **Добавить триггеры свойств:** Добавьте выделенную разметку в ControlTemplate.Triggers блок:

XAML

<ControlTemplate.Triggers>

<!-- Set properties when mouse pointer is over the button. --> <Trigger Property="IsMouseOver" Value="True"> <!-- Below are three property settings that occur when the condition is met (user mouses over button). --> <!-- Change the color of the outer rectangle when user mouses over it. --> <Setter Property ="Rectangle.Stroke" TargetName="outerRectangle" Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}" /> <!-- Sets the glass opacity to 1, therefore, the glass "appears" when user mouses over it. --> <Setter Property="Rectangle.Opacity" Value="1" TargetName="glassCube" /> <!-- Makes the text slightly blurry as though you were looking at it through blurry glass. --> <Setter Property="ContentPresenter.BitmapEffect" TargetName="myContentPresenter"> <Setter.Value> <BlurBitmapEffect Radius="1" /> </Setter.Value> </Setter> </Trigger>

<ControlTemplate.Triggers/>

Нажмите клавишу F5, чтобы запустить приложение и увидеть результат при выполнении указателя мыши над кнопками.

1. **Добавить триггер фокуса:** Далее мы добавим несколько аналогичных методов задания, которые будут обработаны, когда кнопка будет иметь фокус (например, после того, как пользователь щелкнет его).

XAML

<ControlTemplate.Triggers>

<!-- Set properties when mouse pointer is over the button. -->

<Trigger Property="IsMouseOver" Value="True">

<!-- Below are three property settings that occur when the

condition is met (user mouses over button). -->

<!-- Change the color of the outer rectangle when user mouses over it. -->

<Setter Property ="Rectangle.Stroke" TargetName="outerRectangle"

Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}" />

<!-- Sets the glass opacity to 1, therefore, the glass "appears" when user mouses over it. -->

<Setter Property="Rectangle.Opacity" Value="1" TargetName="glassCube" />

<!-- Makes the text slightly blurry as though you were looking at it through blurry glass. -->

<Setter Property="ContentPresenter.BitmapEffect" TargetName="myContentPresenter">

<Setter.Value>

<BlurBitmapEffect Radius="1" />

</Setter.Value>

</Setter>

</Trigger>

<!-- Set properties when button has focus. --> <Trigger Property="IsFocused" Value="true"> <Setter Property="Rectangle.Opacity" Value="1" TargetName="glassCube" /> <Setter Property="Rectangle.Stroke" TargetName="outerRectangle" Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}" /> <Setter Property="Rectangle.Opacity" Value="1" TargetName="glassCube" /> </Trigger>

</ControlTemplate.Triggers>

Нажмите клавишу F5, чтобы запустить приложение, и нажмите одну из кнопок. Обратите внимание, что кнопка остается выделенной после ее нажатия, так как она по-прежнему имеет фокус. Если щелкнуть другую кнопку, кнопка Создать получит фокус, а последний теряет ее.

1. **Добавить анимацию для** MouseEnter **и** MouseLeave**:** Далее мы добавим несколько анимаций к триггерам. Добавьте следующую разметку в любое место внутри ControlTemplate.Triggers блока.

XAML

<!-- Animations that start when mouse enters and leaves button. -->

<EventTrigger RoutedEvent="Mouse.MouseEnter">

<EventTrigger.Actions>

<BeginStoryboard Name="mouseEnterBeginStoryboard">

<Storyboard>

<!-- This animation makes the glass rectangle shrink in the X direction. -->

<DoubleAnimation Storyboard.TargetName="glassCube"

Storyboard.TargetProperty=

"(Rectangle.RenderTransform).(TransformGroup.Children)[0].(ScaleTransform.ScaleX)"

By="-0.1" Duration="0:0:0.5" />

<!-- This animation makes the glass rectangle shrink in the Y direction. -->

<DoubleAnimation

Storyboard.TargetName="glassCube"

Storyboard.TargetProperty=

"(Rectangle.RenderTransform).(TransformGroup.Children)[0].(ScaleTransform.ScaleY)"

By="-0.1" Duration="0:0:0.5" />

</Storyboard>

</BeginStoryboard>

</EventTrigger.Actions>

</EventTrigger>

<EventTrigger RoutedEvent="Mouse.MouseLeave">

<EventTrigger.Actions>

<!-- Stopping the storyboard sets all animated properties back to default. -->

<StopStoryboard BeginStoryboardName="mouseEnterBeginStoryboard" />

</EventTrigger.Actions>

</EventTrigger>

Прозрачный прямоугольник сжимается при наведении указателя мыши на кнопку и возвращается к нормальному размеру, когда указатель покидает.

Существует две анимации, которые запускаются при наведении указателя мыши на кнопку ( MouseEnter событие вызывается). Эти анимации сжимают стеклянный прямоугольник вдоль осей X и Y. Обратите внимание на свойства DoubleAnimation элементов — Duration и By . DurationУказывает, что анимация выполняется в течение половины секунды и By указывает, что стекло сжимается на 10%.

Второй триггер события ( MouseLeave ) просто останавливает первый из них. Когда вы останавливаете Storyboard , все анимированные свойства возвращают значения по умолчанию. Таким образом, когда пользователь перемещает указатель мыши за пределы кнопки, кнопка возвращается к способу, который находился перед перемещением курсора по кнопке.

1. **Добавить анимацию при нажатии кнопки:** Последним шагом является добавление триггера для, когда пользователь нажимает кнопку. Добавьте следующую разметку в ControlTemplate.Triggers любом месте блока:

XAM

<!-- Animation fires when button is clicked, causing glass to spin. -->

<EventTrigger RoutedEvent="Button.Click">

<EventTrigger.Actions>

<BeginStoryboard>

<Storyboard>

<DoubleAnimation Storyboard.TargetName="glassCube"

Storyboard.TargetProperty=

"(Rectangle.RenderTransform).(TransformGroup.Children)[1].(RotateTransform.Angle)"

By="360" Duration="0:0:0.5" />

</Storyboard>

</BeginStoryboard>

</EventTrigger.Actions>

</EventTrigger>

Нажмите клавишу F5, чтобы запустить приложение, и нажмите одну из кнопок. При нажатии кнопки прозрачный прямоугольник вращается вокруг.