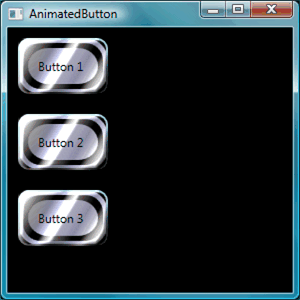
**Пошаговое руководство. Создание кнопки с помощью xaml**

* .NET Framework 4
* .NET Framework 3.5

Цель данного пошагового руководства – ознакомление со способами создания анимированной кнопки в приложении WPF. В данном руководстве используется стиль и шаблон для создания ресурса пользовательской кнопки, которые позволяют повторно использовать код и разделять логику от объявления кнопки. Данное пошаговое руководство приведено полностью в Язык XAML

На следующем рисунке показаны созданные кнопки.



Создание простых кнопок

https://studfile.net/html/2706/309/html_koVRyaTBU4.WSeV/img-EECWyA.png

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

Чтобы создать новый проект wpf и добавить кнопки в окно

1. Запустите Visual Studio.
2. Создайте новый проект WPF: в меню Файл выберите команду Создать и выберите пункт Проект. Перейдите к шаблону Windows Application (WPF) и назовите проект "AnimatedButton". Будет создан скелет для приложения.
3. Добавьте основные кнопки по умолчанию: все необходимы файлы для данного пошагового руководства предоставляются шаблоном. Откройте файл Window1.xaml, дважды щелкнув его в обозревателе решений. По умолчанию в Window1.xaml имеется элемент Grid. Удалите элемент Grid и добавьте несколько кнопок на страницу Язык XAML (Extensible Application Markup Language) путем ввода или копирования и вставки следующего выделенного кода в Window1.xaml:

Копировать

<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>

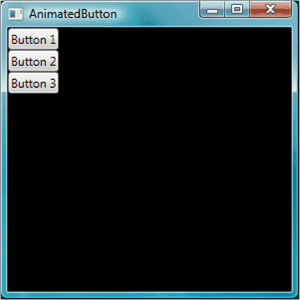
<Button>Button 2</Button>

<Button>Button 3</Button>

</StackPanel>

</Window>

Нажмите клавишу F5 для запуска приложения. Должен появиться набор кнопок, как показано на следующем рисунке.



Теперь, когда простые кнопки созданы, работа в файле Window1.xaml закончена. Остальная часть руководства посвящена файлу App.XAML, определяющему стили и шаблон для кнопок.

Установка основных свойств

https://studfile.net/html/2706/309/html_koVRyaTBU4.WSeV/img-ZN2X0g.png

Теперь зададим некоторые свойства этих кнопок для управления внешним видом и структурой кнопки. Вместо задания свойств для каждой отдельной кнопки, следует использовать ресурсы для определения свойств кнопки для всего приложения. Ресурсы приложений похожи на внешние Спецификация каскадных таблиц стилей (CSS) для веб-страниц. Однако ресурсы являются гораздо более мощным средством, чем Спецификация каскадных таблиц стилей (CSS), что будет продемонстрировано в этом пошаговом руководстве.

# Чтобы использовать стили для установки основных свойств кнопок

## Определите блок Application.Resources: откройте файл app.Xaml и добавьте следующую выделенную разметку (если она еще не добавлена).

Копировать

<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 ресурса определяется по тому, где определяется ресурс.Определение ресурсов в Application.Resoureses в файле app.xaml позволяет использовать ресурсы в любом месте приложения.Подробные сведения об определении области ресурсов см. в разделе Ресурсы XAML.

1. Создайте стиль и задайте значения для его основных свойств: добавьте в блок Application.Resources следующую разметку. Эта разметка создает Style, который применяется для всех кнопок в приложении, устанавливая Width кнопок на 90 и Margin на 10:

Копировать

<Application.Resources>

<Style TargetType="Button">

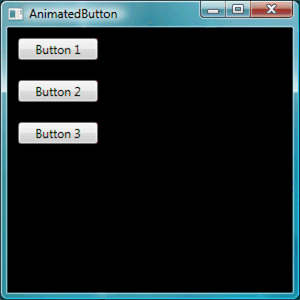
<Setter Property="Width" Value="90" />

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

</Style>

</Application.Resources>

Свойство TargetType указывает, что стиль применяется ко всем объектам типа Button. Каждый Setter устанавливает различное значение свойства для Style. Поэтому на данном этапе каждая кнопка в приложении имеет ширину 90 и поле 10. Если нажать клавишу F5 для запуска приложения, появится следующее окно.



Есть много вариантов работы со стилем, включая различные способы настройки целевых объектов, например, указание сложных значений свойств и даже использование стилей в качестве входных данных для других стилей.

1. Задайте значение свойства стиля для ресурса: Ресурсы предоставляют простой способ повторного использования часто определяемых объектов и значений. Определение сложных значений с помощью ресурсов позволяет сделать код более удобным. Добавьте следующую выделенную разметку в файл app.xaml.

Копировать

<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" />

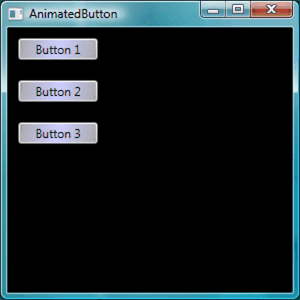
<Setter Property="Margin" Value="10" />

</Style>

</Application.Resources>

Непосредственно под блоком Application.Resources создается ресурс "GrayBlueGradientBrush".Этот ресурс определяет горизонтальный градиент. Этот ресурс может использоваться в качестве значения свойства из любого места в приложении, в том числе из метода задания стиля кнопки для свойства Background. Теперь все кнопки имеют значение свойства Background для этого градиента.

Нажмите клавишу F5 для запуска приложения. Код должен выглядеть следующим образом.



Создание шаблона, который определяет вид элемента Button

https://studfile.net/html/2706/309/html_koVRyaTBU4.WSeV/img-dma3p9.png

В этом разделе создается шаблон, который настраивает внешний вид (представление) кнопки. Представление кнопки состоит из нескольких объектов, включая прямоугольники и другие компоненты, придающих кнопке уникальный вид.

До сих пор управление видом кнопок в приложении выполнялось с помощью изменения свойств кнопки. Что делать, если необходимо кардинально изменить внешний вид кнопки? Шаблоны являются мощным средством управления представлением объекта. Поскольку шаблоны можно использовать в стилях, можно применить шаблон ко всем объектам, к которым применяется стиль (в данном руководстве — кнопка).

# Чтобы использовать шаблон для определения внешнего вида кнопки

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

Копировать

<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. Измените представление кнопки: на данном этапе необходимо определить шаблон. Добавьте следующую выделенную разметку. Эта разметка задает два элемента Rectangle со скругленными краями, за которыми следует DockPanel. DockPanel используется для размещения ContentPresenter кнопки. ContentPresenter отображает содержимое кнопки. В данном пошаговом руководстве содержимым является текст ("Button 1", "Button 2", "Button 3"). Все компоненты шаблона (прямоугольники и DockPanel) располагаются внутри элемента Grid.

Копировать

<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" />

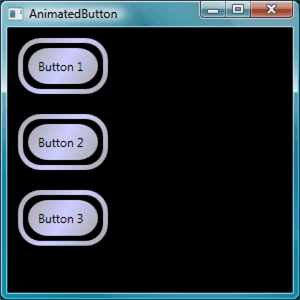
</DockPanel>

</Grid>

</ControlTemplate>

</Setter.Value>

Нажмите клавишу F5 для запуска приложения.Код должен выглядеть следующим образом.



1. Добавьте в шаблон эффект прозрачности: теперь предстоит добавить эффект прозрачности. Сначала создайте ресурсы, которые создают эффект градиента прозрачности. Добавьте эти градиентные ресурсы в любое место внутри блока Application.Resources:

Копировать

<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 как Fill для прямоугольника, который вставляется в Grid шаблона кнопки.Добавьте следующую выделенную разметку в шаблон.

Копировать

<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" />

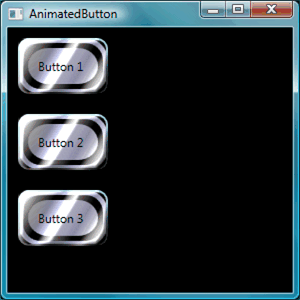
</DockPanel>

</Grid>

</ControlTemplate>

</Setter.Value>

Обратите внимание, что Opacity прямоугольника со свойством x:Name для "glassCube" должна быть равной 0, чтобы при запуске примера не было видно прозрачного прямоугольника.Это необходимо выполнить, поскольку впоследствии будет происходить добавление триггеров в шаблон для взаимодействия пользователя с кнопкой.Однако можно просмотреть, как теперь выглядит кнопка, изменив значение Opacity на 1 и запустив приложение.С м. следующий рисунок. Перед переходом к следующему действию измените Opacity назад на 0.



Создание интерактивной кнопки

https://studfile.net/html/2706/309/html_koVRyaTBU4.WSeV/img-n3Gs7J.png

В этом разделе предстоит создать триггеры свойств и триггеры событий для изменения значений свойств и запуска эффектов анимации в ответ на действия пользователя, например наведение указателя мыши на кнопку и ее нажатие.

Легкий способ сделать кнопку интерактивной (при наведении указателя мыши на кнопку, выходе указателя мыши за границы кнопки, щелчке кнопки мыши и т. д.) является определение триггеров в шаблоне или стиле. Чтобы создать Trigger, определите "условие" для свойства, например: значение свойства IsMouseOver кнопки равно true. Затем предстоит определить элементы Setter (действия), которые выполняются, если условия триггера удовлетворяются.

# Создать интерактивную кнопку

1. Добавьте триггеры шаблона: разделы выделенной разметки в шаблоне.

Копировать

<Setter.Value>

<ControlTemplate TargetType="{x:Кнопка типа}">

<Grid Width="{Ширина шаблонаBinding}"

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

<!-- Внешний прямоугольник с закругленными углами. -->

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

ВертикальноеВыравнивание="Растянуть" Обводка="{TemplateBinding Background}"

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

<!-- Внутренний прямоугольник с закругленными углами. -->

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

ВертикальноеВыравнивание="Растянуть" Обводка="Прозрачный"

ОбводкаТолщина="20"

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

/>

<!-- Стеклянный прямоугольник -->

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

ВертикальноеВыравнивание="Растянуть"

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

Fill="{StaticResource MyGlassBrushResource}"

RenderTransformOrigin="0.5,0.5">

<Прямоугольник.Обводка>

<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>

</Прямоугольник.Обводка>

<!-- Эти преобразования не имеют никакого эффекта, поскольку

здесь объявлены.

Причина включения преобразований заключается в том, что они являются целями.

для анимации (см. ниже). -->

<Rectangle.RenderTransform>

<Трансформгруппа>

<МасштабноеПреобразование />

<РотатеПренсформ />

</ТрансформГрупп>

</Rectangle.RenderTransform>

<!-- Применяется BevelBitmapEffect, чтобы придать кнопке

«Скошенный» вид. -->

<Прямоугольник.BitmapEffect>

<BevelBitmapEffect />

</Rectangle.BitmapEffect>

</Прямоугольник>

<!-- Текущий текст кнопки. -->

<DockPanel Name="myContentPresenterDockPanel">

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

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

</DockPanel>

</Сетка>

<ControlTemplate.Triggers>

<!-- Установите триггеры действий для кнопок и определите

что кнопка делает в ответ на эти триггеры. -->

</ControlTemplate.Triggers>

</ControlTemplate>

</Setter.Value>

1. Добавьте свойства триггеров: разделы выделенную разметку в блоке ControlTemplate.Triggers.

Копировать

<ControlTemplate.Triggers>

<!-- Установите свойства, когда указатель мыши находится над кнопкой. -->

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

<!-- Ниже приведены три настройки свойства, которые возникают, когда

условие выполнено (пользователь наводит курсор мыши на кнопку). -->

<!-- Изменить цвет внешнего прямоугольника, когда пользователь

мышки над ним. -->

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

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

<!-- Устанавливает непрозрачность стекла на 1, поэтому

стекло «появляется», когда пользователь наводит на него курсор мыши. -->

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

<!-- Делает текст слегка размытым, как будто вы

смотрели на него сквозь размытое стекло. -->

<Setter Property="ContentPresenter.BitmapEffect"

TargetName="myContentPresenter">

<Setter.Value>

<BlurBitmapEffect Радиус="1" />

</Setter.Value>

</Сеттер>

</Триггер>

<ControlTemplate.Triggers/>

Нажмите F5, чтобы запустить приложение и увидеть эффект при наведении указателя мыши на кнопку.

1. Добавлен триггер фокуса: теперь добавлены некоторые аналогичные элементы Setter для обработки в случае, когда кнопка не имеет фокуса (например, после того, как пользователь нажмет ее).

Копировать

<ControlTemplate.Triggers>

<!-- Установите свойства, когда указатель мыши находится над кнопкой. -->

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

<!-- Ниже приведены три настройки свойства, которые возникают, когда

условие выполнено (пользователь наводит курсор мыши на кнопку). -->

<!-- Измените цвет внешнего прямоугольника, когда пользователь наводит на него курсор мыши. -->

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

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

<!-- Устанавливает непрозрачность стекла на 1, поэтому стекло «появляется», когда пользователь наводит на него курсор мыши. -->

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

<!-- Делает текст слегка размытым, как будто вы смотрите на него через размытое стекло. -->

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

<Setter.Value>

<BlurBitmapEffect Радиус="1" />

</Setter.Value>

</Сеттер>

</Триггер>

<!-- Установите свойства, когда кнопка находится в фокусе. -->

<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" />

</Триггер>

</ControlTemplate.Triggers>

Нажмите F5, чтобы запустить приложение, а затем нажмите одну из кнопок.Обратите внимание, что кнопка остается выделенной после нажатия, поскольку она по-прежнему имеет фокус.

1. Добавьте анимацию для MouseEnter и MouseLeave :. Далее будет добавлена ​​анимация к триггерам. Добавьте раз эту метку в любое место в блоке ControlTemplate.Triggers.

Копировать

<!-- Анимации, которые запускаются, когда мышь входит и отпускает кнопку. -->

<EventTrigger RoutedEvent="Mouse.MouseEnter">

<EventTrigger.Действия>

<BeginStoryboard Name="mouseEnterBeginStoryboard">

<Раскадровка>

<!-- Эта анимация заставляет стеклянный прямоугольник сжиматься в направлении X. -->

<DoubleAnimation Storyboard.TargetName="glassCube"

Storyboard.TargetProperty=

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

По="-0,1" Продолжительность="0:0:0,5" />

<!-- Эта анимация заставляет стеклянный прямоугольник сжиматься в направлении Y. -->

<Двойная анимация

Storyboard.TargetName="glassCube"

Storyboard.TargetProperty=

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

По="-0,1" Продолжительность="0:0:0,5" />

</Раскадровка>

</Начать раскадровку>

</EventTrigger.Действия>

</EventTrigger>

<EventTrigger RoutedEvent="Mouse.MouseLeave">

<EventTrigger.Действия>

<!-- Остановка раскадровки возвращает все свойства анимации к значениям по умолчанию. -->

<StopStoryboard BeginStoryboardName="mouseEnterBeginStoryboard" />

</EventTrigger.Действия>

</EventTrigger>

Прозрачный эффект, когда указатель мыши наведен на кнопку и возвращается в исходное состояние, при выводе указателя мыши за пределы кнопки.

Создайте две анимации, которые запускаются при наведении указателя мыши на кнопку (вызывается событие MouseEnter ). Эти анимации сжимают контурные контуры по осям X и Y. Обратите внимание на элементы DoubleAnimation - Duration и By . Длительность указывает, что анимация происходит через полсекунды, а указывает , что прозрачный элемент сжимается на 10%.

Триггер второго события ( MouseLeave ) просто останавливается первым из них. При остановке раскадровки все анимированные свойства возвращаются в значениях по умолчанию. об анимации см. в разделе Общие сведения об эффектах анимации .

1. Добавить анимацию при нажатии кнопки: следствием является добавление триггера для событий, которые происходят, когда пользователь нажимает кнопку. Добавьте раз эту метку в любое место в блоке ControlTemplate.Triggers.

Копировать

<!-- Анимация срабатывает при нажатии кнопки, заставляя стекло вращаться. -->

<EventTrigger RoutedEvent="Button.Click">

<EventTrigger.Действия>

<Начать раскадровку>

<Раскадровка>

<DoubleAnimation Storyboard.TargetName="glassCube"

Storyboard.TargetProperty=

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

По="360" Продолжительность="0:0:0.5" />

</Раскадровка>

</Начать раскадровку>

</EventTrigger.Действия>

</EventTrigger>

Нажмите F5, чтобы запустить приложение, а затем нажмите одну из кнопок.

Сводка

https://studfile.net/html/2706/309/html_koVRyaTBU4.WSeV/img-a1hOOL.png

В данном пошаговом руководстве были выполнены следующие упражнения:

* Привязка стиля к типу объекта ( Кнопка ).
* Управление возможностями применения функций во всех приложениях с помощью Style .
* Создание ресурсов, таких как градиенты, для использования в качестве свойств для методов установки Style .
* Настройте внешний вид кнопок во всем приложении с помощью применения к кнопкам шаблона.
* Настройка поведения кнопок в ответ на действия пользователя (такие как MouseEnter , MouseLeave и Click ), включая эффекты анимации.