# Intro

## Intro

<!--x:Class="WPFApplication.MainWindow" - определение класса с кодом окна

Подключения пространств имен для создания элементов управления в разметке.

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

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

<Window x:Class="WPFApplication.MainWindow"

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

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

Title="Главное окно"

Height="350"

Width="525">

<Window.Background>

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

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

<GradientStop Color="#FFA3BB0C" Offset="1"/>

</LinearGradientBrush>

</Window.Background>

<Grid>

<!--Блок текста-->

<TextBlock Width="100"

Height="24"

Margin="210,93,207,202"

Foreground="Red"

Text="Введите свое имя">

</TextBlock>

<!--Поле ввода-->

<TextBox Name="textBoxName"

Width="100"

Height="24"

Margin="202,139,201,148" Text="" AcceptsTab="False">

</TextBox>

<!--Кнопка-->

<Button Width="100"

Height="24"

Content="Ok"

Margin="202,182,200,105"

Click="Button\_Click">

<Button.Background>

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

<GradientStop Color="#FFF3F3F3" Offset="0"/>

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

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

<GradientStop Color="#FF80D639" Offset="1"/>

</LinearGradientBrush>

</Button.Background>

</Button>

</Grid>

</Window>

## Gradient Rectangle

<Window x:Class="PropertiesInMarkUp.MainWindow"

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

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

Title="Установка свойств в разметке" Height="280" Width="525">

<StackPanel>

<!--Установка значения свойства Fill через атрибут-->

<Rectangle Fill="Green" Width="100" Height="100">

</Rectangle>

<!--Установка значения свойства Fill через вложенный элемент-->

<Rectangle Width="100" Height="100" Margin="10">

<Rectangle.Fill>

<LinearGradientBrush>

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

<GradientStop Color="#FFE2B82B" Offset="1"></GradientStop>

</LinearGradientBrush>

</Rectangle.Fill>

</Rectangle>

</StackPanel>

</Window>

## Events

<Window x:Class="EventsInMarkup.MainWindow"

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

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

Title="Установка события в разметке. События мыши." Height="350" Width="525">

<StackPanel>

<!--Прямоугольникам установлены обработчики на события мыши:

MouseEnter - курсор попал в область прямоугольника.

MouseLeave - курсор вышел за область прямоугольника. -->

<Rectangle MinHeight="100" MinWidth="100" Fill="Green" Margin="10"

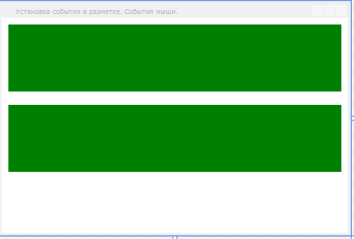
MouseEnter="Rectangle\_MouseEnter" MouseLeave="Rectangle\_MouseLeave"></Rectangle>

<Rectangle MinHeight="100" MinWidth="100" Fill="Green" Margin="10"

MouseEnter="Rectangle\_MouseEnter" MouseLeave="Rectangle\_MouseLeave"></Rectangle>

</StackPanel>

</Window>



## Grid

<Window x:Class="\_8Ball.MainWindow"

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

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

Title="Магический 8 Ball" Height="350" Width="525">

<!--

Объект Grid Определяет гибкую область сетки, состоящую из столбцов и строк.

-->

<Grid>

<!--

RowDefinitions - Коллекция которая определяет

строки для отображения - RowDefinition.

-->

<Grid.RowDefinitions>

<!--

Создаем три строки(RowDefinition) в коллекции - RowDefinitions.

(Height="100\*"):

100 - Фиксированные размер.

100\* - Указывает, что начальный размер ряда значение до звздочки, но размер может пропорционально менятся при изменении размера окна.

Auto - Указывает что размер данного элемента будет подобран под размер

самого большого объекта в контейнере.

-->

<RowDefinition Height="100\*" />

<RowDefinition Height="Auto" />

<RowDefinition Height="100\*" />

</Grid.RowDefinitions>

<!--

Используем сложную кисть для заливки фона градиентом.

Добавляем для этого дескриптор - Background.

-->

<Grid.Background>

<!--

LinearGradientBrush - Заполняет область линейным градиентом.

Линейный градиент определяет градиент вдоль прямой линии.

Конечные точки линии определяются свойствами StartPoint и EndPoint

линейного градиента.

Кисть LinearGradientBrush рисует свои GradientStops вдоль этой линии.

-->

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

<!-- Коллекция свойств GradientStop -->

<LinearGradientBrush.GradientStops>

<!--

GradientStop имеет свойства:

Offset - Задает позици цвета.

Color - Задает цвет.

-->

<GradientStop Offset="1" Color="#FFEEEEEE" />

<GradientStop Offset="0.881" Color="#FF302F2F" />

</LinearGradientBrush.GradientStops>

</LinearGradientBrush>

</Grid.Background>

<!--

<TextBox Имя объкта.

Name="txtQuestion"

Выравнивание элемента по горизонтали относительно контейнера.

HorizontalAlignment="Stretch"

Выравнивание элемента по вертикали относительно контейнера.

VerticalAlignment="Stretch"

Выравнивание элемента относительно сторон контейнера.

Margin="10"

Пренос на новую строку в случае если текст не помещается в одной строке.

TextWrapping="Wrap"

Шрифт который будем использовать.

FontFamily="Verdana"

Размер используемого шрифта.

FontSize="20"

Строка в контейнере в которой будет находится данный элемент.

Grid.Row="0"

Цвет используемого шрифта.

Foreground="#FF646464"

Свойство текст.

Text="[Введите свой вопрос.]" />

-->

<TextBox Name="txtQuestion"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"

Margin="10"

TextWrapping="Wrap"

FontFamily="Verdana"

FontSize="20"

Grid.Row="0"

Foreground="#FF646464"

Text="[Введите свой вопрос.]" />

<!--

IsDefault - Получает или задает значение, указывающее, является ли Button

кнопкой по умолчанию.

-->

<Button HorizontalAlignment="Left"

VerticalAlignment="Center"

Margin="10,0,10,0"

Padding="5"

Grid.Row="1"

Content="Ответ на вопрос"

Click="Answer\_Click"

IsDefault="True" />

<TextBox Name="txtAnswer"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"

Margin="10"

TextWrapping="Wrap"

FontFamily="Verdana"

FontSize="20"

IsReadOnly="True"

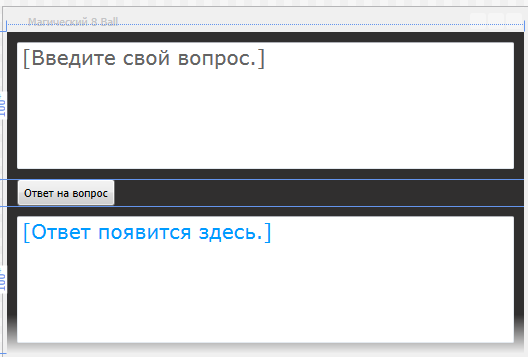
Grid.Row="2"

Foreground="#FF009BFF"

Text="[Ответ появится здесь.]" />

</Grid>

</Window>



# Layout

## Grid

<Grid ShowGridLines="True">

<!--Определение колонок таблицы-->

<Grid.ColumnDefinitions>

<ColumnDefinition></ColumnDefinition>

<ColumnDefinition></ColumnDefinition>

</Grid.ColumnDefinitions>

<!--Определение строк таблицы-->

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<TextBlock Grid.Row="0" Grid.Column="0"

Padding="10" Margin="20" Background="LightCoral">

Grid.Row="0" Grid.Column="0"</TextBlock>

<TextBlock Grid.Row="0" Grid.Column="1"

Padding="10" Margin="20" Background="LightCyan">

Grid.Row="0" Grid.Column="1"</TextBlock>

<TextBlock Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"

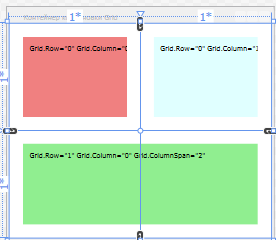
Padding="10" Margin="20" Background="LightGreen">

Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"

</TextBlock>

</Grid>

</Window>



## Grid size

<Grid>

<Grid.RowDefinitions>

<!--Высота строки 100-->

<RowDefinition Height="100"></RowDefinition>

<!--Высота строки изменяется пропорционально с изменением окна-->

<RowDefinition Height="\*"></RowDefinition>

<!--Высота равна высоте самого большого элемента определенного в строке-->

<RowDefinition Height="Auto"></RowDefinition>

</Grid.RowDefinitions>

<TextBlock Grid.Row="0"

Background="LightBlue" Width="1000">TextBlock 1 (Height="100")</TextBlock>

<TextBlock Grid.Row="1"

Background="LightGreen">TextBlock 2 (Height="\*")</TextBlock>

<TextBlock Grid.Row="2"

Background="LightCyan" Height="50">TextBlock 3 (Height="Auto")</TextBlock>

</Grid>

</Window>

## StackPanel

<!--StackPanel - Располагает дочерние элементы в одну строку, которую можно ориентировать по горизонтали или по вертикали -->

<StackPanel Orientation="Horizontal">

<TextBlock Background="LightGoldenrodYellow" Padding="10" Margin="10">TextBlock 1</TextBlock>

<TextBlock Background="LightBlue" Padding="10" Margin="10">TextBlock 2</TextBlock>

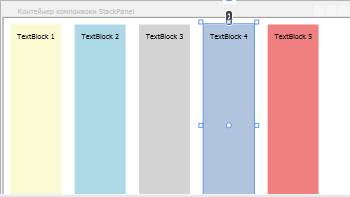
<TextBlock Background="LightGray" Padding="10" Margin="10">TextBlock 3</TextBlock>

<TextBlock Background="LightSteelBlue" Padding="10" Margin="10">TextBlock 4</TextBlock>

<TextBlock Background="LightCoral" Padding="10" Margin="10">TextBlock 5</TextBlock>

</StackPanel>

</Window>



## DockPanel

<!--

DockPanel - определяет область, в которой можно упорядочить дочерние элементы

горизонтально или вертикально относительно друг друга.

DockPanel.Dock - указывает, к какой стороне контейнера следует привязать контрол.

-->

<DockPanel LastChildFill="False">

<TextBlock DockPanel.Dock="Top"

Background="LightBlue"

Padding="10">

DockPanel.Dock="Top"</TextBlock>

<TextBlock DockPanel.Dock="Top"

Background="Aqua"

Padding="10">

Toolbar</TextBlock>

<TextBlock DockPanel.Dock="Bottom"

Background="LightGreen"

Padding="10">

DockPanel.Dock="Bottom"</TextBlock>

<TextBlock DockPanel.Dock="Right"

Background="LightCyan"

Padding="10">

DockPanel.Dock="Right"</TextBlock>

<TextBlock DockPanel.Dock="Left"

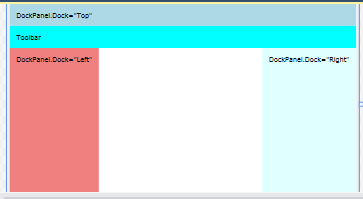
Background="LightCoral"

Padding="10">

DockPanel.Dock="Left"</TextBlock>

</DockPanel>

</Window>



## WrapPanel

<!--WrapPanel - Размещает дочерние элементы последовательно слева направо,

разбивая содержимое до следующей строки на краю содержащего окна.

Последующее размещение происходит последовательно сверху вниз или справа налево,

в зависимости от значения свойства Orientation. -->

<WrapPanel Orientation="Horizontal">

<TextBlock Background="LightGoldenrodYellow" Padding="10" Margin="10">TextBlock 1</TextBlock>

<TextBlock Background="LightBlue" Padding="10" Margin="10">TextBlock 2</TextBlock>

<TextBlock Background="LightGray" Padding="10" Margin="10">TextBlock 3</TextBlock>

<TextBlock Background="LightSteelBlue" Padding="10" Margin="10">TextBlock 4</TextBlock>

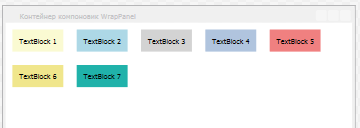
<TextBlock Background="LightCoral" Padding="10" Margin="10">TextBlock 5</TextBlock>

<TextBlock Background="Khaki" Padding="10" Margin="10">TextBlock 6</TextBlock>

<TextBlock Background="LightSeaGreen" Padding="10" Margin="10">TextBlock 7</TextBlock>

</WrapPanel>

</Window>



## UniformGrid

<!--UniformGrid - Предоставляет способ размещения элементов в сетки,

при котором все ячейки имеют одинаковый размер.-->

<UniformGrid Rows="2" Columns="2">

<TextBlock Background="LightGoldenrodYellow" Padding="10" Margin="10">TextBlock 1</TextBlock>

<TextBlock Background="LightBlue" Padding="10" Margin="10">TextBlock 2</TextBlock>

<TextBlock Background="LightGray" Padding="10" Margin="10">TextBlock 3</TextBlock>

<TextBlock Background="LightSteelBlue" Padding="10" Margin="10">TextBlock 4</TextBlock>

</UniformGrid>

</Window>



## Canvas

<!--Canvas - Определяет область, в рамках которой можно явно расположить дочерние элементы

путем использования координат, являющихся относительными к области Canvas.-->

<Canvas>

<TextBlock Canvas.Left="30" Canvas.Top="60"

Height="100" Background="LightCoral"

TextWrapping="WrapWithOverflow" Padding="10">

Canvas.Left="30" Canvas.Top="60"

</TextBlock>

<TextBlock Canvas.Left="160" Canvas.Top="120"

Height="100" Width="200" Background="LightGreen"

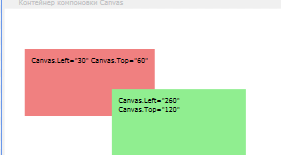
TextWrapping="WrapWithOverflow" Padding="10">

Canvas.Left="260" Canvas.Top="120"

</TextBlock>

</Canvas>

</Window>



## IncCanvas

<Window x:Class="\_07\_IncCanvas.MainWindow"

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

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

Title="Контейнер компоновик InkCanvas" Height="350" Width="525" Loaded="Window\_Loaded">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition></RowDefinition>

<RowDefinition Height="22"></RowDefinition>

</Grid.RowDefinitions>

<StackPanel Margin="5" Orientation="Horizontal">

<TextBlock Margin="5" Text="Режим редактирования: " />

<ComboBox Name="lstEditingMode"

VerticalAlignment="Center"

HorizontalAlignment="Stretch"

Margin="0,0,10,0" />

</StackPanel>

<!--

InkCanvas - Определяет область, которая рисует и отображает рукописные штрихи.

-->

<InkCanvas Name="inkCanvas"

Grid.Row="1"

Background="LightBlue"

EditingMode="{Binding ElementName=lstEditingMode, Path=SelectedItem}"

Gesture="inkCanvas\_Gesture">

<Button InkCanvas.Left="10" InkCanvas.Top="10">Button</Button>

<TextBox InkCanvas.Left="100" InkCanvas.Top="32">TextBox</TextBox>

</InkCanvas>

<TextBox Name="gestureName"

HorizontalAlignment="Stretch"

VerticalAlignment="Bottom"

Grid.Row="2" />

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void inkCanvas\_Gesture(object sender, InkCanvasGestureEventArgs e)

{

String gestures = "";

// Выборка "предпологаемых" гестур.

foreach (GestureRecognitionResult res in e.GetGestureRecognitionResults())

gestures += res.ApplicationGesture.ToString() + "; ";

// Отображаем название гестуры.

gestureName.Text = gestures;

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

// Выборка всех режимов редактирования InkCanvas.

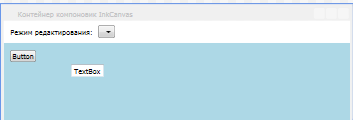
foreach (InkCanvasEditingMode mode in Enum.GetValues(typeof(InkCanvasEditingMode)))

lstEditingMode.Items.Add(mode);

lstEditingMode.SelectedItem = inkCanvas.EditingMode;

}

}



## GridSplitter

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<TextBlock Grid.Row="0" Grid.Column="0" Background="LightGoldenrodYellow" Padding="10">

TextBlock 1</TextBlock>

<TextBlock Grid.Row="1" Grid.Column="0" Background="LightGray" Padding="10">

TextBlock 2</TextBlock>

<GridSplitter Grid.Row="0" Grid.Column="1" Grid.RowSpan="2" Width="3"

HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Background="Black"></GridSplitter>

<!--Вложенный Grid-->

<Grid Grid.Row="0" Grid.Column="2" Grid.RowSpan="2">

<Grid.RowDefinitions>

<RowDefinition Height="\*"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="\*"></RowDefinition>

</Grid.RowDefinitions>

<TextBlock Grid.Row="0" Grid.Column="0" Background="LightGreen" Padding="10">

TextBlock 1</TextBlock>

<TextBlock Grid.Row="2" Grid.Column="0" Background="LightBlue" Padding="10">

TextBlock 2</TextBlock>

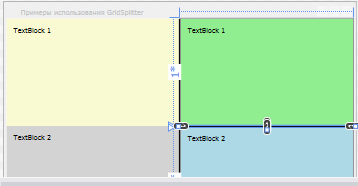
<GridSplitter Grid.Row="1" Grid.Column="0" Height="3"

HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Background="Black"></GridSplitter>

</Grid>

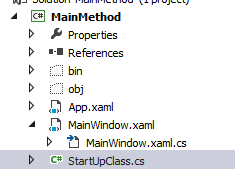
</Grid>

</Window>



# Application

## Main Method



namespace ApplicationSample

{

// В проекте созданном по стандартному шаблону WPF Application нет метода Main, так как он

// генерируется автоматически.

// Найти авто сгенерированный файл можно по пути /obj/x86/Debug/App.g.cs

// Если требуется создать свой метод Main, следует указать в настройках проекта,

// что стартовым объектом (StartUp Object) является Ваш класс.

// Это можно сделать в окне Properties во вкладке Application.

public class StartUpClass

{

[STAThread]

static void Main(string[] args)

{

Application app = new Application();

MainWindow window = new MainWindow();

app.Run(window);

}

}

}

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

protected override void OnStartup(StartupEventArgs e)

{

Thread.Sleep(1000);

base.OnStartup(e);

}

}

}

## Events

App.xaml

public partial class App : Application

{

public App()

{

this.Startup += new StartupEventHandler(App\_Startup);

this.Exit += new ExitEventHandler(App\_Exit);

this.SessionEnding += new SessionEndingCancelEventHandler(App\_SessionEnding);

this.Activated += new EventHandler(App\_Activated);

this.Deactivated += new EventHandler(App\_Deactivated);

this.DispatcherUnhandledException += new DispatcherUnhandledExceptionEventHandler(App\_DispatcherUnhandledException);

}

protected override void OnActivated(EventArgs e)

{

Debug.WriteLine("------> Activated");

base.OnActivated(e);

}

void App\_Startup(object sender, StartupEventArgs e)

{

// Событие происходит посел запуска метода Run и до появления главного окна.

Debug.WriteLine("------> Startup");

}

void App\_Activated(object sender, EventArgs e)

{

// Присходит когда активизируется одно из окон приложения.

// Например, при переходе с друго приложения запущенного в системе.

Debug.WriteLine("------> Activated");

}

void App\_Deactivated(object sender, EventArgs e)

{

// Присходит при деактивации окна приложения. Например, при переключении на другое окно.

Debug.WriteLine("------> Deactivated");

}

void App\_SessionEnding(object sender, SessionEndingCancelEventArgs e)

{

// Присходит когда завершаеся сеанс Windows

Debug.WriteLine("------> SessionEnding " + e.ReasonSessionEnding);

e.Cancel = true; // предотвращаем завершение выполнения.

}

void App\_Exit(object sender, ExitEventArgs e)

{

// Присходит когда приложение закрывается до того как метод Run вернет управление.

Debug.WriteLine("------> Exit");

}

void App\_DispatcherUnhandledException(object sender, DispatcherUnhandledExceptionEventArgs e)

{

// Присходит когда в главном потоке приложения происходит необработанное исключение.

Debug.WriteLine("------> DispatcherUnhandledException");

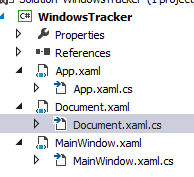
e.Handled = true; // помечаем необработанное исключение, как обработанное.

}

}

}

## Document



/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

private List<Document> documents = new List<Document>();

public List<Document> Documents

{

get { return documents; }

set { documents = value; }

}

}

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void cmdCreate\_Click(object sender, RoutedEventArgs e)

{

// Создаем окно

Document doc = new Document();

// Устанавливаем владельца для созданного окна

doc.Owner = this;

// Отображаем окно.

doc.Show();

// Добавляем окно в коллекцию окон.

(Application.Current as App).Documents.Add(doc);

}

private void cmdUpdate\_Click(object sender, RoutedEventArgs e)

{

// Перебираем коллекцию окон

foreach (Document doc in ((App)Application.Current).Documents)

{

// Обновляем содержимое всех окно.

doc.SetContent("Updated at " + DateTime.Now.ToLongTimeString());

}

}

}

public partial class Document : Window

{

public Document()

{

InitializeComponent();

}

public void SetContent(string content)

{

this.Content = content;

}

private void Window\_Closed(object sender, EventArgs e)

{

(Application.Current as App).Documents.Remove(this);

}

}

## Instance Application

namespace SingleInstanceApplication

{

public class StartupClass

{

[STAThread]

static void Main(string[] args)

{

SingleInstanceApplicationWrapper startWrapper = new SingleInstanceApplicationWrapper();

startWrapper.Run(args);

}

}

// WindowsFormsApplicationBase из сборки Microsoft.VisualBasic

public class SingleInstanceApplicationWrapper : Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase

{

private WpfApplication \_app;

public SingleInstanceApplicationWrapper()

{

// Включаем режим single-instance.

this.IsSingleInstance = true;

}

// Первый запуск приложения.

protected override bool OnStartup(Microsoft.VisualBasic.ApplicationServices.StartupEventArgs eventArgs)

{

try

{

// Регестрация расширения .test. Нужно запустить приложения с правами администратора.

string extension = ".test";

string title = "SingleInstanceApplication";

string extensionDescription = "A Test Document";

ExstensionRegisterHelper.SetFileAssociation(extension, title + "." + extensionDescription);

}

catch

{

MessageBox.Show("Не удалось зарегистрировать расширение .test");

}

\_app = new WpfApplication();

\_app.Run();

return false;

}

// Метод срабатывает при последующих запусках приложения.

protected override void OnStartupNextInstance(Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs eventArgs)

{

if (eventArgs.CommandLine.Count > 0)

{

(Application.Current.MainWindow as MainWindow).ShowFileText(eventArgs.CommandLine[0]);

}

Application.Current.MainWindow.Activate();

}

}

class WpfApplication : Application

{

protected override void OnStartup(StartupEventArgs e)

{

base.OnStartup(e);

// загружаем главное окно.

MainWindow window = new MainWindow();

this.MainWindow = window;

window.Show();

if (e.Args.Length > 0)

{

ShowDocument(e.Args[0]);

}

}

private void ShowDocument(string path)

{

(this.MainWindow as MainWindow).ShowFileText(path);

}

}

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

public void ShowFileText(string path)

{

string fileContent = File.ReadAllText(path);

this.Content = fileContent;

}

}

/// <summary>

/// Класс для регистрации рсширения файла .test для открытия его в текущем приложении при двойном клике.

/// </summary>

class ExstensionRegisterHelper

{

// Значения

// extension = .test

// progID = SingleInstanceApplication.A Test Document

public static void SetFileAssociation(string extension, string progID)

{

// Создание ключа в реестре .testDoc

SetValue(Registry.ClassesRoot, extension, progID);

// Узнаем где находится в данный момент сборка.

string assemblyFullPath = System.Reflection.Assembly.GetExecutingAssembly().Location.Replace("/", @"\");

StringBuilder sbShellEntry = new StringBuilder();

sbShellEntry.AppendFormat("\"{0}\" \"%1\"", assemblyFullPath);

// Создаем в реестре ключ SingleInstanceApplication.A Test Document\shell\open\command для определения приложения которое должно запускать формат .testDoc

SetValue(Registry.ClassesRoot, progID + @"\shell\open\command", sbShellEntry.ToString());

StringBuilder sbDefaultIconEntry = new StringBuilder();

sbDefaultIconEntry.AppendFormat("\"{0}\",0", assemblyFullPath);

// Создаем в реестре ключ SingleInstanceApplication.A Test Document\DefaultIcon для

SetValue(Registry.ClassesRoot, progID + @"\DefaultIcon", sbDefaultIconEntry.ToString());

// Create application subkey

SetValue(Registry.ClassesRoot, @"Applications\" + Path.GetFileName(assemblyFullPath), "", "NoOpenWith");

}

private static void SetValue(RegistryKey root, string subKey, object keyValue)

{

SetValue(root, subKey, keyValue, null);

}

private static void SetValue(RegistryKey root, string subKey, object keyValue, string valueName)

{

bool hasSubKey = ((subKey != null) && (subKey.Length > 0));

RegistryKey key = root;

try

{

if (hasSubKey) key = root.CreateSubKey(subKey);

key.SetValue(valueName, keyValue);

}

finally

{

if (hasSubKey && (key != null)) key.Close();

}

}

## Multi Thread

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void buttonNewThread\_Click(object sender, RoutedEventArgs e)

{

//Thread.Sleep(TimeSpan.FromSeconds(5));

// Создание вторичного потока.

Thread th = new Thread(UpdateTextRight);

th.Start();

MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString());

}

private void UpdateTextRight()

{

// Задержка на 5 секунд.

Thread.Sleep(TimeSpan.FromSeconds(5));

// v1.0

ThreadStart threadStart = new ThreadStart(WorkerMethod);

this.Dispatcher.BeginInvoke(DispatcherPriority.Normal, threadStart);

// v2.0

txt.Dispatcher.BeginInvoke(DispatcherPriority.Normal, (Action)delegate() { txt.Text = "Text"; });

}

/// v1.0

private void WorkerMethod()

{

MessageBox.Show(Thread.CurrentThread.GetHashCode().ToString());

// Этот метод выполняется в потоке диспетчера, поэтому трудоемкие операции подвесят приложение

// точно так же, как если бы оно работало в одном потоке.

//Thread.Sleep(TimeSpan.FromSeconds(5));

txt.Text = "Test";

}

}

## Background Worker

<Window x:Class="BackgroundWorkerSample.MainWindow"

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

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

Title="MainWindow" Height="107" Width="525">

<Grid>

<ProgressBar Height="10" HorizontalAlignment="Left" Margin="12,12,0,0" Name="progressBar1" VerticalAlignment="Top" Width="479" Maximum="100" />

<Button Content="Start Background Worker" Height="23" HorizontalAlignment="Left" Margin="12,30,0,0" Name="buttonStartWorker" VerticalAlignment="Top" Width="410" Click="buttonStartWorker\_Click" />

<Button Content="Cancel" Height="23" HorizontalAlignment="Left" Margin="428,30,0,0" Name="buttonCancel" VerticalAlignment="Top" Width="63" Click="buttonCancel\_Click" />

</Grid>

</Window>

public partial class MainWindow : Window

{

// Задача объекта типа BackgroundWorker захватить свободный поток из пула потоков CLR и затем из

// этого потока вызвать событие DoWork;

BackgroundWorker \_worker;

public MainWindow()

{

InitializeComponent();

\_worker = new BackgroundWorker();

// Метод, который будет выполнятся в отдельном потоке. Событие DoWork срабатывает при вызове RunWorkerAsync

\_worker.DoWork += new DoWorkEventHandler(worker\_DoWork);

// Метод, который сработает в момент завершения BackgroundWorker

\_worker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(worker\_RunWorkerCompleted);

// Событие для отслеживание процесса выполнения задачи BackgroundWorker. Событие возникает при вызове метода \_worker.ReportProgress(i);

\_worker.ProgressChanged += new ProgressChangedEventHandler(worker\_ProgressChanged);

// Для отслеживания выполнения хода работ свойство WorkerReportsProgress устанавливаем true

\_worker.WorkerReportsProgress = true;

// Поддержка отмены выполнения фоновой операции с помощью метода CancelAsync()

\_worker.WorkerSupportsCancellation = true;

}

// Метод работает из потока Dispatcher. Он может получать доступ к переменным окна.

void worker\_ProgressChanged(object sender, ProgressChangedEventArgs e)

{

progressBar1.Value = e.ProgressPercentage;

}

// Метод работает из потока Dispetcher. Он может получать доступ к переменным окна.

void worker\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

this.Title = "Completed";

MessageBox.Show("Completed");

if (e.Cancelled)

this.Title = "Cancelled";

}

// Данный метод работает в отдельном потоке.

void worker\_DoWork(object sender, DoWorkEventArgs e)

{

for (int i = 0; i <= 100; ++i)

{

// Эмулируем трудоемкую задачу.

Thread.Sleep(50);

// Отмена выполнения фоновой задачи, сработает при вызове CancelAsync

if (\_worker.CancellationPending)

{

e.Cancel = true; // значение нужно установить для того что бы при событии RunWorkerCompleted определить почему оно было вызвано, из-за того что закончилась операция или из-за отмены.

return; // Отмена выполнения фоновой операции.

}

// Отчитываемся о проценте выполнения задачи.

\_worker.ReportProgress(i);

}

}

private void buttonStartWorker\_Click(object sender, RoutedEventArgs e)

{

// Запуск выполнения фоновой операции. Событие DoWork.

// Вторая перегрузка RunWorkerAsync позволяет передать объект событию DoWork для его последующей обработки в потоке.

\_worker.RunWorkerAsync();

}

private void buttonCancel\_Click(object sender, RoutedEventArgs e)

{

// Для работы метода, свойство WorkerSupportsCancellation должно быть равное true.

\_worker.CancelAsync();

}

}

# Content

## Content

<Window x:Class="ContentInWPF.MainWindow"

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

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

Title="Контент в WPF" Height="199" Width="525">

<Grid>

<StackPanel>

<Button Name="button1" Height="80"></Button>

<Button Name="button2" Height="80"></Button>

</StackPanel>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

// Test - не наследуется от UIElement. Используется метод ToString

button1.Content = new Test();

// TextBox - производный от UIElement.

TextBox textBox = new TextBox();

textBox.Width = 100;

button2.Content = textBox;

}

}

class Test

{

public override string ToString()

{

return "Hello world";

}

}

## Dynamic content

<Window x:Class="\_001\_DynamicContent.Window1"

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

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

Title="Window1" SizeToContent="WidthAndHeight">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="\*" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*" />

<ColumnDefinition Width="\*" />

</Grid.ColumnDefinitions>

<!--

StackPanel - размещает в себе элементы друг за другом,

по горизонтали или по вертикали.

-->

<StackPanel Grid.Row="0" Grid.Column="0">

<Button Name="cmdPrev" Margin="10,10,10,3">

Назад

</Button>

<Button Name="cmdNext" Margin="10,3,10,3">

Вперед

</Button>

<CheckBox Name="chkLongText"

Margin="10,10,10,10"

Checked="chkLongText\_Checked"

Unchecked="chkLongText\_Unchecked"

Content="Показать длинный текст" />

<Button Name="cmdClose"

VerticalAlignment="Bottom"

Margin="10"

Content="Закрыть" Click="cmdClose\_Click" />

</StackPanel>

<TextBox Grid.Row="0"

Grid.Column="1"

Margin="0,10,10,10"

TextWrapping="Wrap"

Grid.RowSpan="2"

Background="#22009BFF">

Этот пример демонстрирует, как кнопки адаптируются

под содержимое, которое в них находится. Такое поведение делает

локализацию приложения намного проще.

</TextBox>

</Grid>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

}

private void chkLongText\_Checked(object sender, RoutedEventArgs e)

{

cmdPrev.Content = " <- Перейти к предыдущему окну - Window ";

cmdNext.Content = " Перейти к следующему окну - Window -> ";

}

private void chkLongText\_Unchecked(object sender, RoutedEventArgs e)

{

cmdPrev.Content = "Назад";

cmdNext.Content = "Вперед";

}

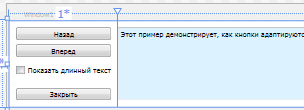
private void cmdClose\_Click(object sender, RoutedEventArgs e)

{

Application.Current.Shutdown();

}

}



## Modular content

<Window x:Class="\_002\_ModularContent.Window1"

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

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

Title="Window1"

Height="292"

Width="452">

<!--

ScrollViewer - отображает полосы прокрутки, если контент не помещается в заданную область.

-->

<ScrollViewer>

<StackPanel>

<WrapPanel Background="LightSteelBlue" Name="pnlList">

<CheckBox Margin="5" IsChecked="True">Panel1</CheckBox>

<CheckBox Margin="5" IsChecked="True">Panel2</CheckBox>

<CheckBox Margin="5" IsChecked="True">Panel3</CheckBox>

<CheckBox Margin="5" IsChecked="True">Panel4</CheckBox>

</WrapPanel>

<WrapPanel Name="panel">

<StackPanel Name="Panel1">

<Border Padding="5" BorderBrush="Yellow" BorderThickness="2">

<UniformGrid Rows="2" Columns="2">

<Button Margin="10" Padding="10">1</Button>

<Button Margin="10" Padding="10">2</Button>

<Button Margin="10" Padding="10">3</Button>

<Button Margin="10" Padding="10">4</Button>

</UniformGrid>

</Border>

</StackPanel>

<StackPanel Name="Panel2">

<Border Padding="15" BorderBrush="Yellow" BorderThickness="2">

<TabControl >

<TabItem Header="Page1">

<Button Padding="100,50,100,50">Tabs</Button>

</TabItem>

<TabItem Header="Page2">

</TabItem>

</TabControl>

</Border>

</StackPanel>

<StackPanel Name="Panel3">

<Border Padding="15" BorderBrush="Yellow" BorderThickness="2">

<StackPanel>

<TextBox MinLines="5" MaxWidth="150" TextWrapping="Wrap">This is a test of a text box that contains wrapped text.</TextBox>

<StackPanel Orientation="Horizontal">

<Button>OK</Button>

<Button>Cancel</Button>

</StackPanel>

</StackPanel>

</Border>

</StackPanel>

<StackPanel Name="Panel4">

<Border Padding="15" BorderBrush="Yellow" BorderThickness="2">

<UniformGrid Rows="2" Columns="2">

<Button Margin="10" Padding="10">1</Button>

<Button Margin="10" Padding="10">2</Button>

<Button Margin="10" Padding="10">3</Button>

<Button Margin="10" Padding="10">4</Button>

</UniformGrid>

</Border>

</StackPanel>

</WrapPanel>

</StackPanel>

</ScrollViewer>

</

Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

// Вешаем обработчик на события (Checked и Unchecked) всех CheckBox(ов), которые находятся внутри окна.

AddHandler(CheckBox.CheckedEvent, new RoutedEventHandler(chk\_Checked));

AddHandler(CheckBox.UncheckedEvent, new RoutedEventHandler(chk\_Unchecked));

}

private void chk\_Checked(object sender, RoutedEventArgs e)

{

// Копируем ссылку на используемый CheckBox.

// OriginalSource - свойство содержащее отправителя события.

CheckBox chk = e.OriginalSource as CheckBox;

// При помощи системного класса LogicalTreeHelper и его метода FindLogicalNode(),

// можно выполнить поиск какого либо элемента в XAML коде элемента переданого в аргументы по имени.

DependencyObject dpObj = LogicalTreeHelper.FindLogicalNode(panel, chk.Content.ToString());

// Показываем элемент который мы получили. (Panel1, Panel2, и т.д.)

((FrameworkElement)dpObj).Visibility = Visibility.Visible;

}

private void chk\_Unchecked(object sender, RoutedEventArgs e)

{

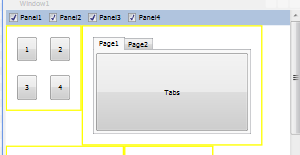
CheckBox chk = e.OriginalSource as CheckBox;

DependencyObject obj = LogicalTreeHelper.FindLogicalNode(panel, chk.Content.ToString());

((FrameworkElement)obj).Visibility = Visibility.Collapsed;

}

}



## Property content

<Window x:Class="\_003\_ProprtyContent.Window1"

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

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

Title="Главное окно"

Height="310"

Width="227">

<!--

Тип свойства Content - object. В качестве значения этому свойству можно передать любой объект.

Элементы управления, которые имеют свойство текст, могут содержать в себе текст, изображения, другие

элементы управления и т.д.

-->

<StackPanel>

<Button Name="button1" Margin="3">

<StackPanel Orientation="Horizontal">

<Image Source="icon1.png" Width="24" Height="24" />

<Label Content="Простая кнопка" />

</StackPanel>

</Button>

<StackPanel Orientation="Horizontal">

<Button Height="64" Name="button2" Margin="3">

<Image Source="icon1.png" />

</Button>

<Button Height="64" Name="button3" Margin="3">

<Image Source="icon2.png" />

</Button>

<Button Height="64" Name="button4" Margin="3">

<Image Source="icon3.png" />

</Button>

</StackPanel>

<StackPanel Orientation="Horizontal">

<Button Name="button5" Margin="3">

<StackPanel>

<Image Source="icon1.png" Width="58" />

<TextBlock HorizontalAlignment="Center" Text="К 1" />

</StackPanel>

</Button>

<Button Name="button6" Margin="3">

<StackPanel>

<Image Source="icon2.png" Width="58" />

<Label HorizontalAlignment="Center" Content="К 2" />

</StackPanel>

</Button>

<Button Name="button7" Margin="3">

<StackPanel>

<Image Source="icon3.png" Width="58" />

<Label HorizontalAlignment="Center" Content="К 3" />

</StackPanel>

</Button>

</StackPanel>

<Button Name="button8" Margin="3">

<StackPanel Orientation="Horizontal">

<Image Source="icon4.png" Width="58" />

<StackPanel>

<Label Content="Введите свое имя:" />

<TextBox />

</StackPanel>

</StackPanel>

</Button>

</StackPanel>

</Window>



## ScrollViewer

<Window x:Class="\_004\_ScrollViewer.Window1"

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

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

Title="Window1"

Height="213"

Width="300">

<DockPanel>

<StackPanel Orientation="Horizontal"

HorizontalAlignment="Center"

DockPanel.Dock="Top"

Margin="0,2,0,2">

<Button Name="button1" Click="button1\_Click" ToolTip="Line down">

<Image Source="down.png" Width="32" />

</Button>

<Button Name="button2" Click="button2\_Click" ToolTip="Line up">

<Image Source="up.png" Width="32" />

</Button>

<Separator Margin="64,0,64,0" />

<Button Name="button3" Click="button3\_Click" ToolTip="Page down">

<Image Source="page\_down.png" Width="32" />

</Button>

<Button Name="button4" Click="button4\_Click" ToolTip="Page up">

<Image Source="page\_up.png" Width="32" />

</Button>

</StackPanel>

<!--

ScrollViewer - Представляет прокручиваемую область,

в которой могут содержаться другие видимые элементы.

-->

<ScrollViewer Name="scroller">

<Grid Margin="0,10,0,0" Focusable="False">

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="Auto" />

<ColumnDefinition Width="\*" MinWidth="110" />

<ColumnDefinition Width="Auto" />

</Grid.ColumnDefinitions>

<Label Grid.Row="0"

Grid.Column="0"

Margin="3"

VerticalAlignment="Center"

Content="Home:" />

<TextBox Grid.Row="0"

Grid.Column="1"

Margin="3"

Height="Auto"

VerticalAlignment="Center" />

<Label Grid.Row="1"

Grid.Column="0"

Margin="3"

VerticalAlignment="Center"

Content="Network:" />

<TextBox Grid.Row="1"

Grid.Column="1"

Margin="3"

Height="Auto"

VerticalAlignment="Center" />

<Label Grid.Row="2"

Grid.Column="0"

Margin="3"

VerticalAlignment="Center"

Content="Network:" />

<TextBox Grid.Row="2"

Grid.Column="1"

Margin="3"

Height="Auto"

VerticalAlignment="Center" />

<Label Grid.Row="3"

Grid.Column="0"

Margin="3"

VerticalAlignment="Center"

Content="Network:" />

<TextBox Grid.Row="3"

Grid.Column="1"

Margin="3"

Height="Auto"

VerticalAlignment="Center" />

<Label Grid.Row="4"

Grid.Column="0"

Margin="3"

VerticalAlignment="Center"

Content="Network:" />

<TextBox Grid.Row="4"

Grid.Column="1"

Margin="3"

Height="Auto"

VerticalAlignment="Center" />

<Label Grid.Row="5"

Grid.Column="0"

Margin="3"

VerticalAlignment="Center"

Content="Network:" />

<TextBox Grid.Row="5"

Grid.Column="1"

Margin="3"

Height="Auto"

VerticalAlignment="Center" />

</Grid>

</ScrollViewer>

</DockPanel>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

// Прокручивает содержимое на одну строку вверх.

scroller.LineUp();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

// Прокручивает содержимое на одну строку вниз.

scroller.LineDown();

}

private void button4\_Click(object sender, RoutedEventArgs e)

{

// Прокручивает содержимое на одну страницу вверх.

scroller.PageUp();

}

private void button3\_Click(object sender, RoutedEventArgs e)

{

// Прокручивает содержимое на одну страницу вниз.

scroller.PageDown();

}

}

## GroupBox

<Window x:Class="\_005\_GroupBox.Window1"

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

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

Title="GroupBox Sample"

Height="177"

Width="207">

<Grid>

<!--

GroupBox - используется для группирования элементов управления.

Контролы RadioButton групируются, когда находятся в одном контейнере компоновки.

-->

<GroupBox Header="GroupBox" Margin="10" VerticalAlignment="Top">

<StackPanel>

<RadioButton Content="Radio button 1" Margin="3" />

<RadioButton Content="Radio button 2" Margin="3" />

<RadioButton Content="Radio button 3" Margin="3" />

<Button Width="100" HorizontalAlignment="Left" Content="Применить" Margin="3" Padding="3" />

</StackPanel>

</GroupBox>

</Grid>

</Window>

## TabControl1

<Window x:Class="\_006\_TabControl.Window1"

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

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

Title="TabControl Samples"

Height="300"

Width="300">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="135\*" />

<RowDefinition Height="127\*" />

</Grid.RowDefinitions>

<TabControl Name="tabControl1">

<TabItem Header="Category 1" IsSelected="True">

<StackPanel>

<TextBox />

<Button Content="Button" />

</StackPanel>

</TabItem>

<TabItem Header="Category 2">

<Image Source="image.png" Stretch="None" />

</TabItem>

<TabItem Header="Category 3" />

</TabControl>

<TabControl TabStripPlacement="Bottom" Grid.Row="1">

<TabItem Header="Item 1" IsSelected="True" />

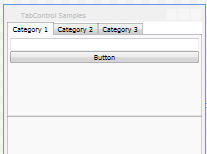
<TabItem Header="Item 2" />

<TabItem Header="Item 3" />

</TabControl>

</Grid>

</Window>



## TabControl2

<Window x:Class="TabControl\_2.MainWindow"

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

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

Title="TabControl Samples" Height="350" Width="525">

<Grid>

<TabControl HorizontalAlignment="Stretch" Margin="5" Name="tabControl1" VerticalAlignment="Stretch">

<TabItem Name="tabItem1">

<TabItem.Header>

<Image Source="image.png" Width="40"></Image>

</TabItem.Header>

<Grid />

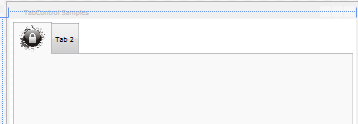
</TabItem>

<TabItem Header="Tab 2" />

</TabControl>

</Grid>

</Window>



## Expander

<Window x:Class="\_007\_Expander.Window1"

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

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

Title="Expander Samples" Height="355" Width="330">

<StackPanel>

<!--

Expander - Представляет элемент управления, отображающий

заголовок со свертываемым окном, отображающим содержимое.

-->

<Expander Header="Expander 1"

IsExpanded="False"

Margin="5"

Padding="3"

BorderBrush="Black">

<Button Content="Button" />

</Expander>

<Expander Header="Expander 2"

IsExpanded="True"

MaxHeight="200"

Margin="5"

Padding="3"

BorderBrush="Black">

<TextBox TextWrapping="Wrap" IsReadOnly="True" VerticalScrollBarVisibility="Auto">

Windows Presentation Foundation (WPF, кодовое название — Avalon)

— графическая (презентационная) подсистема в составе

.NET Framework 3.0, имеющая прямое отношение к XAML. WPF вместе с

.NET Framework 3.0 предустановлена в Windows Vista, а также

доступна для установки в Windows XP SP2 и Windows Server 2003.

Это первое реальное обновление технологической среды пользовательского

интерфейса со времени выпуска Windows 95. Оно включает новое ядро,

которое должно заменить GDI и GDI+, используемые в нынешней Windows-платформе.

WPF представляет собой высокоуровневый объектно-ориентированный функциональный

слой (framework), позволяющий создавать 2D- и 3D-интерфейсы. Сейчас его можно

назвать альтернативным вариантом Adobe Flash и Java-апплетам, используемым

в Web-разработке применительно к Windows Forms. Но в будущем WPF должен

объединить Windows и Web-разработку (в том числе AJAX).

</TextBox>

</Expander>

<Expander ExpandDirection="Right"

Margin="5"

Padding="3"

Header="Expander 3"

IsExpanded="False"

BorderBrush="Black">

<Button Height="23">Test Button</Button>

</Expander>

</StackPanel>

</Window>

## Decorator Border

<Window x:Class="DecoratorBorder.MainWindow"

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

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

Title="Decorator Border" Height="350" Width="525">

<Grid>

<Border Background="LightBlue"

BorderBrush="Gray"

BorderThickness="3"

CornerRadius="50 50 40 40"

Padding="15"

Margin="15">

<Button Name="button1" Height="25" Width="140" Click="button1\_Click">Click me</Button>

</Border>

</Grid>

</Window>

# Dependency properties and Routed events

## Dependency

<Window x:Class="WpfDependency.MainWindow"

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

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

Title="Dependency Property Samples" Height="350" Width="525"

xmlns:myControls="clr-namespace:WpfDependency">

<Grid>

<!--Выражение привязки {Binding ....} может быть присвоено только свойству зависимостей-->

<myControls:MyFirstControl

x:Name="myFirstControl1"

Data="{Binding ElementName=slider1, Path=Value}">

</myControls:MyFirstControl>

<Button Name="button1"

Content="Get Value"

Height="23"

HorizontalAlignment="Left"

Margin="12,276,0,0"

VerticalAlignment="Top"

Width="75"

Click="button1\_Click" />

<Slider Name="slider1"

Height="23"

HorizontalAlignment="Left"

Margin="30,37,0,0"

VerticalAlignment="Top"

Width="437" />

</Grid>

</Window>

class MyFirstControl : FrameworkElement

{

// 1. Создание свойства зависимостей.

// Идентификатор свойства зависимости - поле представляющее свойство зависимости.

public static DependencyProperty DataProperty;

// 2. регистрация свойства зависимостей

static MyFirstControl()

{

// параметр 1: Имя свойства.

// параметр 2: Тип данных свойства.

// параметр 3: Тип, которому принадлежит это свойство.

DataProperty = DependencyProperty.Register("Data", typeof(int), typeof(MyFirstControl));

}

// 3. Упаковка свойства зависимостей в традиционное свойство.

// Методы SetValue и GetValue унаследованы от класса DependencyObject

public int Data

{

get

{

return (int)GetValue(DataProperty);

}

set

{

SetValue(DataProperty, value);

}

}

}

## Dependency property

<Window x:Class="WpfDependencyProperty.MainWindow"

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

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

Title="Dependency Property Samples" Height="139" Width="388">

<Grid>

<StackPanel Margin="249,0,0,0">

<Button Name="button1"

Content="Set Value"

Margin="2"

Click="button1\_Click"

Height="23"

VerticalAlignment="Top" />

<Button Name="button2"

Content="Get Value"

Margin="2"

Click="button2\_Click" />

<Button Name="button3"

Content="Default Value"

Margin="2"

Click="button3\_Click" />

</StackPanel>

<TextBox Name="textBox1"

Height="99"

HorizontalAlignment="Left"

Margin="0,1,0,0"

VerticalAlignment="Top"

Width="243"

Text="Hello world"/>

</Grid>

</Window>

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

// v1.0

// Объект со свойством зависимости.

//SimpleProperty \_testObject = new SimpleProperty();

// v2.0

// Объект со свойством зависимости. Со

//значением по умолчанию и проверкой правильности данных.

PropertyMetadata \_testObject = new PropertyMetadata();

public MainWindow()

{

InitializeComponent();

}

// Установить значение свойству.

private void button1\_Click(object sender, RoutedEventArgs e)

{

\_testObject.TestData = textBox1.Text;

textBox1.Text = string.Empty;

}

// Прочитать значение свойства.

private void button2\_Click(object sender, RoutedEventArgs e)

{

textBox1.Text = \_testObject.TestData;

}

// Очистить значение свойства.

private void button3\_Click(object sender, RoutedEventArgs e)

{

// Если свойство зависимостей создавалось с метаданными, которые указывали значение по умолчанию,

// то вызов ClearValue() установит значение по умолчанию

// v1.0

//\_testObject.ClearValue(SimpleProperty.TestDataProperty);

// v2.0

\_testObject.ClearValue(PropertyMetadata.TestDataProperty);

}

}

class SimpleProperty : DependencyObject

{

// Идентификатор свойства зависимости - поле представляющее свойство зависимости. Согласно условию должно иметь имя плюс слово Property в конце.

// Таким образом можно отделить имя свойства от имени свойства зависимости.

// свойство зависимости обязательно должно быть public и static

public static DependencyProperty TestDataProperty;

// Статический конструктор. Сработает до выполнения любого кода данного экземпляра.

static SimpleProperty()

{

// Регистрация свойства зависимости. Это нужно сделать до того, как свойство начнет использоваться в коде.

TestDataProperty = DependencyProperty.Register("TestData", typeof(string), typeof(SimpleProperty));

// параметр 1: Имя свойства.

// параметр 2: Тип данных свойства.

// параметр 3: Тип, которому принадлежит это свойство.

}

// Упаковка свойства зависимостей в традиционное свойство.

// Методы SetValue и GetValue унаследованы от класса DependencyObject

public string TestData

{

set

{

SetValue(TestDataProperty, value);

}

get

{

return (string)GetValue(TestDataProperty);

}

}

}

// Свойство зависимостей, которое использует метаданные

class PropertyMetadata : DependencyObject

{

// Поле представляющее свойство зависимости. Согласно условию должно иметь имя плюс слово Property в конце.

// Таким образом, можно отделить имя свойства от имени свойства зависимости.

public static readonly DependencyProperty TestDataProperty;

// Статический конструктор. Сработает до выполнения любого кода данного экземпляра.

static PropertyMetadata()

{

// Объект указывает какие службы вы хотите использовать вместе со своим свойством зависимостей (например, поддержку привязки данных, анимации и т.д.)

FrameworkPropertyMetadata metadata = new FrameworkPropertyMetadata("default value", FrameworkPropertyMetadataOptions.NotDataBindable);

// Регистрация свойства зависимости. Это нужно сделать до того, как свойство начнет использоваться в коде.

TestDataProperty = DependencyProperty.Register("TestData", typeof(string), typeof(PropertyMetadata), metadata, new ValidateValueCallback(Validate));

// параметр 1: Имя свойства.

// параметр 2: Тип данных свойства.

// параметр 3: Тип, которому принадлежит это свойство.

// параметр 4: Метаданные.

// параметр 5: Метод для проверки корректности значения.

}

// Упаковка свойства зависимостей в традиционное свойство.

// Методы SetValue и GetValue унаследованы от класса DependencyObject

public string TestData

{

set { SetValue(TestDataProperty, value); }

get { return (string)GetValue(TestDataProperty); }

}

// Проверка значения.

static bool Validate(object value)

{

if (value.ToString() == "error")

return false;

return true;

}

}

## DependencyPropCallBack

<Window x:Class="WpfDependencyPropCallback.MainWindow"

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

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

Height="238" Width="431"

xmlns:my="clr-namespace:WpfDependencyPropCallback"> <!--Подключение пространства имен-->

<Grid>

<Button Content="Get Property" Height="23" HorizontalAlignment="Left" Margin="95,42,0,0" Name="button1" VerticalAlignment="Top" Width="90" Click="button1\_Click" />

<TextBox Height="23" HorizontalAlignment="Left" Margin="191,42,0,0" Name="textBox1" VerticalAlignment="Top" Width="120" />

<my:SimplePropClass x:Name="simplePropClass1" MyData="{Binding ElementName=slider1, Path=Value}"></my:SimplePropClass>

<Slider Height="23" HorizontalAlignment="Left" Margin="30,12,0,0" Name="slider1" VerticalAlignment="Top" Width="354" Maximum="150"/>

<TextBlock TextWrapping="Wrap" Margin="15" Padding="7" Name="textBlock1" VerticalAlignment="Bottom">

Свойство Value, элемента управление Slider, связано со свойством MyData объекта SimplePropClass, который создан в XAML коде. Если

свойство MyData будет обычным свойством, а не свойством зависимостей, то привязка данных работать не будет.

</TextBlock>

</Grid>

</Window>

public partial class MainWindow : Window

{

SimplePropClass \_testObject = new SimplePropClass();

public MainWindow()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

textBox1.Text = simplePropClass1.MyData.ToString();

}

}

// Свойство зависимостей и CallBack методы проверяющие значение установленное свойству.

class SimplePropClass : FrameworkElement

{

// Метаданные.

static FrameworkPropertyMetadata metadata = new FrameworkPropertyMetadata(

new PropertyChangedCallback(ChangedCallbackMethod), new CoerceValueCallback(CoerceValueCallbackMethod));

// Свойство зависимостей.

public static readonly DependencyProperty MyDataProperty =

DependencyProperty.Register("MyData",

typeof(int),

typeof(SimplePropClass),

metadata,

new ValidateValueCallback(ValidateValueCallbackMethod));

// Свойство-обертка для свойства зависимостей.

public int MyData

{

get { return (int)GetValue(MyDataProperty); }

set

{

SetValue(MyDataProperty, value);

// В методах доступа нет смысла вставлять свой код, так как они используются только на этапе проектирования.

// При присвоении значений из XAML кода, данные будут записаны непосредственно в DependencyProperty минуя этот код.

MessageBox.Show("Hello");

}

}

// Срабатывает первым.

// Метод, который будет срабатывать при обновлении значения свойства для корректирования значения если оно не подходит. (Указывается через метаданные).

static object CoerceValueCallbackMethod(DependencyObject d, object baseValue)

{

if ((int)baseValue <= 100)

return baseValue;

return 100;

}

// Срабатывает вторым.

// Проверка на валидность введенных данных. (Привязывается при регистрации свойства).

static bool ValidateValueCallbackMethod(object value)

{

if ((int)value < 0)

return false;

return true;

}

// Срабатывает третьим.

// Метод который будет срабатывать при обновлении значения свойства. (Указывается через метаданные)

static void ChangedCallbackMethod(DependencyObject d, DependencyPropertyChangedEventArgs e)

{

Application.Current.MainWindow.Title = e.NewValue.ToString();

}

}

## Route events

<Window x:Class="WpfRoutedEventSample.MainWindow"

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

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

Title="Routed Events Samples" Height="350" Width="525"

xmlns:myButtons="clr-namespace:WpfRoutedEventSample">

<Grid>

<myButtons:ExtraButton x:Name="OuterButton"

Margin="30"

MyButtonClick="ExtraButton\_MyButtonClick">

<StackPanel>

<myButtons:ExtraButton x:Name="InnerButton"

Padding="30"

MyButtonClick="ExtraButton\_MyButtonClick" Content="Вложенная кнопка">

</myButtons:ExtraButton>

</StackPanel>

</myButtons:ExtraButton>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void ExtraButton\_MyButtonClick(object sender, RoutedEventArgs e)

{

MessageBox.Show((sender as Button).Name);

}

}

class ExtraButton : Button

{

// Маршрутизируемое событие.

public static RoutedEvent MyButtonClickEvent;

// Статический конструктор, в котором регистрируется событие.

static ExtraButton()

{

// Регистрация события с помощью EventManager.

MyButtonClickEvent = EventManager.RegisterRoutedEvent("MyButtonClick",

RoutingStrategy.Tunnel,

typeof(RoutedEventHandler),

typeof(ExtraButton));

// 1 параметр: имя события.

// 2 параметр: тип маршрута. (Поднимающийся, туннельный, прямой)

// 3 параметр: тип делегата, который будет задавать сигнатуру обработчика.

// 4 параметр: класс-владелец события.

#region Типы маршрутов

// RoutingStrategy.Bubble - событие идет от самого последнего (вложенного элемента) до родительского верхнего уровня.

// RoutingStrategy.Tunnel - событие идет от самого верхнего элемента (родительского до дочернего.

// RoutingStrategy.Direct - событие для одного элемента.

#endregion

}

// Обертка для маршрутизируемого события.

public event RoutedEventHandler MyButtonClick

{

add { AddHandler(MyButtonClickEvent, value); }

remove { RemoveHandler(MyButtonClickEvent, value); }

}

// Переопределение метода, который срабатывает при нажатии на кнопку.

protected override void OnClick()

{

base.OnClick();

// Аргумент, который будет передан обработчику события.

RoutedEventArgs args = new RoutedEventArgs(ExtraButton.MyButtonClickEvent, this);

// Вызов события. Событие, которое должно быть вызвано, определяется по параметрам объекта типа RoutedEventArgs

RaiseEvent(args);

}

}

## Bubbled label click

<Window x:Class="\_006\_BubledLabelClick.Window1"

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

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

Title="Маршрутизируемые события"

Height="522"

Width="327">

<Grid MouseUp="MouseUp\_Handler"

Margin="3">

<Grid.RowDefinitions>

<RowDefinition Height="Auto" />

<RowDefinition Height="\*" />

<RowDefinition Height="Auto" />

<RowDefinition Height="Auto" />

</Grid.RowDefinitions>

<Label MouseUp="MouseUp\_Handler"

Margin="5"

Background="AliceBlue"

BorderBrush="Black"

BorderThickness="1"

HorizontalAlignment="Center">

<StackPanel MouseUp="MouseUp\_Handler">

<TextBlock MouseUp="MouseUp\_Handler"

Margin="3"

Text="Рисунок"

HorizontalAlignment="Center" />

<Image MouseUp="MouseUp\_Handler"

Source="happy.gif"

Stretch="None" />

</StackPanel>

</Label>

<ListBox Name="lstMessages"

Margin="5"

Grid.Row="1" />

<CheckBox Name="chkHandle"

Margin="5"

Grid.Row="2"

Content="Событие обработано" />

<Button Click="cmdClear\_Click"

Grid.Row="3"

HorizontalAlignment="Right"

Margin="5"

Padding="3"

Content="Отчистить список" />

</Grid>

</Window>

public partial class Window1 : Window

{

// Счетчик событий.

private Int32 eventCounter = 0;

public Window1()

{

InitializeComponent();

}

private void MouseUp\_Handler(object sender, RoutedEventArgs e)

{

eventCounter++;

// Сбор информации о событии.

string message = "#" + eventCounter.ToString() + ":\r\n" +

" Sender: " + sender.ToString() + "\r\n" +

" Source: " + e.Source + "\r\n" +

" Original Source: " + e.OriginalSource;

// Добавляем сгенерированную строку в ListBox.

lstMessages.Items.Add(message);

// Свойство Handled - Получает или задает значение, указывающее текущее состояние обработки

// маршрутизируемого события по мере продвижения его по маршруту.

e.Handled = (bool)chkHandle.IsChecked;

}

private void cmdClear\_Click(object sender, RoutedEventArgs e)

{

// Сбрасываем счетчик.

eventCounter = 0;

// Чистим ListBox.

lstMessages.Items.Clear();

}

}

## Key press event

<Window x:Class="\_005\_KeyPressEvent.Window1"

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

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

Title="События клавиатуры"

Height="596"

Width="419">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="\*"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

</Grid.RowDefinitions>

<DockPanel Margin="5">

<TextBlock Margin="3" >

Вводите текст:

</TextBlock>

<TextBox PreviewKeyDown="KeyEvent"

KeyDown="KeyEvent"

PreviewKeyUp="KeyEvent"

KeyUp="KeyEvent"

PreviewTextInput="TextInputHandler"

TextChanged="TextChanged">

</TextBox>

</DockPanel>

<ListBox Margin="5" Name="lstMessages" Grid.Row="1"></ListBox>

<Button Click="cmdClear\_Click" Grid.Row="3" HorizontalAlignment="Right" Margin="5" Padding="3">Очистить список</Button>

</Grid>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

}

/\*

События в порядке их возникновения:

1. PreviewKeyDown – нажатие клавиши.

2. KeyDown – нажатие клавиши.

3. PreviewTextInput – нажатие завершено и элемент получил текстовый ввод. (не работает для тех клавиш, которые не отображаются)

4. TextChanged – смена текста в элементе управления.

5. PreviewKeyUp – отпускание клавиши.

6. KeyUp - отпускание клавиши.

Все события Preview нужны для того, что бы отловить события, которые были перехвачены стандартной реализацией элемента управления.

Например, TextBox блокирует клавиши управление курсором, так как обрабатывает их самостоятельно.

\*/

private void KeyEvent(object sender, KeyEventArgs e)

{

string message =

"Event: " + e.RoutedEvent + " " +

" Key: " + e.Key;

lstMessages.Items.Add(message);

}

private void TextInputHandler(object sender, TextCompositionEventArgs e)

{

string message =

"Event: " + e.RoutedEvent + " " +

" Text: " + e.Text;

lstMessages.Items.Add(message);

}

private void TextChanged(object sender, TextChangedEventArgs e)

{

string message =

"Event: " + e.RoutedEvent;

lstMessages.Items.Add(message);

}

private void cmdClear\_Click(object sender, RoutedEventArgs e)

{

lstMessages.Items.Clear();

}

}

## Key modifiers

<Window x:Class="KeyModifiers.MainWindow"

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

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

Title="Shift Ctrl Alt" Height="347" Width="298">

<StackPanel Margin="5">

<TextBox KeyDown="KeyEvent"></TextBox>

<TextBlock Name="lblInfo"></TextBlock>

<Button Click="CheckShift">Проверить состояние клавиши Shift</Button>

</StackPanel>

</Window>

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void KeyEvent(object sender, KeyEventArgs e)

{

// Выводим список всех клавиш, которые в данный момент нажаты, таких как Ctrl, Shift и т.д.

lblInfo.Text = "Modifiers: " +

e.KeyboardDevice.Modifiers.ToString();

// Проверка нажатия Ctrl клавиши.

// e.KeyboardDevice.Modifiers & ModifierKeys.Control - операция нужна для того что бы проверить нажат ли Contrl при зажатии нескольких клавиш.

// см. Определение перечисления ModifierKeys и целочисленные значения, которые в нем используются.

//if ((e.KeyboardDevice.Modifiers & ModifierKeys.Control) == ModifierKeys.Control)

//{

// lblInfo.Text += "\r\nЗажата клавиша Ctrl.";

//}

// 0111

// &

// 0010

// 0010

if ((e.KeyboardDevice.Modifiers.HasFlag(ModifierKeys.Control)))

{

lblInfo.Text += "\r\nЗажата клавиша Ctrl.";

}

}

// Обработчик для кнопки на форме.

private void CheckShift(object sender, RoutedEventArgs e)

{

// Проверка нажата ли клавиша Shift

if (Keyboard.IsKeyDown(Key.LeftShift))

{

lblInfo.Text = "Зажата клавиша Shift.";

}

else

{

lblInfo.Text = "Клавиша Shift не нажата.";

}

}

## Only numbers check

<Window x:Class="\_004\_OnlyNumbers.Window1"

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

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

Title="События клавиатуры. Перехват неправильного ввода."

Height="135"

Width="300">

<StackPanel Margin="10"

PreviewTextInput="pnl\_PreviewTextInput"

PreviewKeyDown="pnl\_PreviewKeyDown">

<TextBox Name="textBox1" Height="22" VerticalAlignment="Top" Margin="2" />

<TextBox Name="textBox2" Height="22" VerticalAlignment="Top" Margin="2" />

<TextBox Name="textBox3" Height="22" VerticalAlignment="Top" Margin="2" />

</StackPanel>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

}

// Событие PreviewTextInput не срабатывает для Contrl, Shift, Alt и Space

// События клавиатуры являются туннельными.

private void pnl\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

short value;

// Если: введена не цифра

if (!Int16.TryParse(e.Text, out value))

{

// То: Указываем, что событие обработано и распространятся далее не должно.

e.Handled = true;

}

}

private void pnl\_PreviewKeyDown(object sender, KeyEventArgs e)

{

if (e.Key == Key.Space)

{

e.Handled = true;

}

}

}

# Classic controls

## ListBox items

<Window x:Class="CheckBoxList.MainWindow"

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

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

Title="CheckBox Samples" Height="350" Width="525">

<Grid Margin="10">

<Grid.RowDefinitions>

<RowDefinition Height="\*"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

</Grid.RowDefinitions>

<!--CheckBox.Click - определение обработчика для всех элементов управления, которые находятся в ListBox

SelectionChanged - обработчик на смену выделения элемента в ListBox

-->

<ListBox Name="lst"

SelectionChanged="lst\_SelectionChanged"

CheckBox.Click="lst\_SelectionChanged">

<CheckBox Margin="3">Option 1</CheckBox>

<CheckBox Margin="3">Option 2</CheckBox>

<CheckBox Margin="3">Option 3</CheckBox>

</ListBox>

<StackPanel Grid.Row="1" Margin="0,10,0,0">

<TextBlock>Текущий выбор:</TextBlock>

<TextBlock Name="txtSelection" TextWrapping="Wrap"></TextBlock>

<Button Margin="0,10,0,0" Click="cmd\_CheckAllItems">Проверить все элементы</Button>

</StackPanel>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void lst\_SelectionChanged(object sender, RoutedEventArgs e)

{

// Если источник события CheckBox

if (e.OriginalSource is CheckBox)

{

// В ListBox выбранным элементом делаем этот CheckBox

lst.SelectedItem = e.OriginalSource;

}

// SelectedIndex - позиция выбранного элемента.

// SelectedItem - объект выбранный в списке.

txtSelection.Text = String.Format("Позиция {0}.\r\nСостояние {1}.",

lst.SelectedIndex,

((CheckBox)lst.SelectedItem).IsChecked);

}

private void cmd\_CheckAllItems(object sender, RoutedEventArgs e)

{

StringBuilder sb = new StringBuilder();

// Проходим по всем элементам в ListBox

foreach (CheckBox item in lst.Items)

{

if (item.IsChecked == true)

{

sb.AppendLine(item.Content + " отмечен. ");

}

}

txtSelection.Text = sb.ToString();

}

}

## Embedded font

<Window x:Class="EmbededFont.MainWindow"

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

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

Title="Встраивание шрифтов в сборку" Height="77" Width="300">

<Grid>

<TextBox FontFamily="./#SNAP ITC" FontSize="20">This is embedded font</TextBox>

</Grid>

</Window>

## Image list

<Window x:Class="\_009\_ImageList.Window1"

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

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

Title="ListBox Sample"

Height="300"

Width="300">

<!--Если в ListBox помещен объект производный от UIElement, то этот элемент визуализируется,

иначе для объекта вызывается метод ToString() и полученная строка выводится как элемент списка

SelectionMode - Режим выбора элементов в списке:

Single - может быть выбран только один элемент.

Multiple - может быть выбрано несколько элементов, при этом для того что бы произвести выбор

или снять выбор нужно сделать левый клик по элементу.

Extended - множественный выбор производится с зажатой клавишей Ctrl или Shift

-->

<ListBox Margin="5"

SelectionMode="Extended"

Name="lst">

<StackPanel Orientation="Horizontal">

<Image Source="1.png" Width="64" Height="64" />

<Label VerticalContentAlignment="Center" Content="Первый рисунок" />

</StackPanel>

<StackPanel Orientation="Horizontal">

<Image Source="2.png" Width="64" Height="64" />

<Label VerticalContentAlignment="Center" Content="Второй рисунок" />

</StackPanel>

<StackPanel Orientation="Horizontal">

<Image Source="3.png" Width="64" Height="64" />

<Label VerticalContentAlignment="Center" Content="Третий рисунок" />

</StackPanel>

</ListBox>

</Window>

## Label

<Window x:Class="\_005\_Label.Window1"

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

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

Title="Label Sample"

Height="190"

Width="300">

<StackPanel Margin="5">

<!--

Label - предоставляет возможность фокусировки на элементах управления после нажатия на горячие клавиши.

Для фокусировки используются свойства Target и Content.

Target – элемент, к которому будет происходить привязка.

Content - текст и горячая клавиша (в комбинации с Alt) формат: "\_x - \_X".

-->

<Label Target="{Binding ElementName=txtA}" Content="Выбор \_X" />

<TextBox Name="txtA" />

<Label Target="{Binding ElementName=txtB}" Content="Выбор \_Y" />

<TextBox Name="txtB" />

<Label HorizontalAlignment="Center"

Margin="10"

Content="Нажмите кнопку alt + &quot;X&quot; или alt + &quot;Y&quot;" />

<!-- &quot; - используется для указания символа ". -->

</StackPanel>

</Window>

## Popup

<Window x:Class="\_007\_Popup.Window1"

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

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

Title="Настройка всплывающей подсказки (Popup)"

Height="300"

Width="300">

<Grid Margin="10">

<TextBlock TextWrapping="Wrap">

Этот текст показывает, как можно в TextBlock создать

<!--Run - декорирует и форматирует текст.-->

<Run TextDecorations="Underline" MouseEnter="run\_MouseEnter">

ссылку

</Run>.

</TextBlock>

<!--

Popup - Всплывающее окно как и ToolTip.

Отличия от ToolTip:

1) Отображается и прячется Popup

2) Имеет свойство PopupAnimation.

3) Popup может принимать фокус, таким образом, в него можно помещать другие элементы управления.

Свойства:

StaysOpen - Оставлять открытым после открытия.

Placement="Mouse" - относительно чего будет положение окна.

PopupAnimation="Slide" - Тип анимация первого отображения.

AllowsTransparency="True" - Поддержка прозрачности.

-->

<Popup Name="popLink"

StaysOpen="False"

Placement="Mouse"

MaxWidth="200"

PopupAnimation="Scroll"

AllowsTransparency="True">

<Border BorderThickness="3"

CornerRadius="3"

BorderBrush="#AA323232"

Background="#AA000000">

<TextBlock Margin="10" TextWrapping="Wrap" Foreground="White">

Ссылка в текстовом блоке.

<!--

Hyperlink - ссылка.

Свойство NavigateUri - содержит ссылку на ресурс.

-->

<Hyperlink Foreground="#009BFF"

NavigateUri="http://en.wikipedia.org/wiki/Term"

Click="lnk\_Click">

Wikipedia

</Hyperlink>

</TextBlock>

</Border>

</Popup>

</Grid>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

}

// Обработчик наведения на объект Run.

private void run\_MouseEnter(object sender, MouseEventArgs e)

{

// Делаем Popup видимым.

popLink.IsOpen = true;

}

// Переход по ссылке.

private void lnk\_Click(object sender, RoutedEventArgs e)

{

string fileName = ((Hyperlink)sender).NavigateUri.ToString();

// Запускаем приложение, которое ассоциируется с файлом.

Process.Start(fileName);

}

}

## Radio button

<Window x:Class="RadioButtonList.MainWindow"

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

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

Title="RadioButton Samples" Height="350" Width="525">

<Grid>

<StackPanel>

<!--RadioButton по умолчанию группируется контейнерами, но также,

можно группировать эти элементы управления с помощью свойства GroupName-->

<GroupBox Margin="5" Padding="10" Header="Radiobutton Group 1">

<StackPanel>

<RadioButton>Group 1</RadioButton>

<RadioButton>Group 1</RadioButton>

<RadioButton>Group 1</RadioButton>

<RadioButton Margin="0,10,0,0"

GroupName="Group2">Group 2</RadioButton>

</StackPanel>

</GroupBox>

<GroupBox Margin="5" Padding="10" Header="Radiobutton Group 2">

<StackPanel>

<RadioButton>Group 3</RadioButton>

<RadioButton>Group 3</RadioButton>

<RadioButton>Group 3</RadioButton>

<RadioButton Margin="0,10,0,0"

GroupName="Group2">Group 2</RadioButton>

</StackPanel>

</GroupBox>

</StackPanel>

</Grid>

</Window>

## Tool tip

<Window x:Class="ToolTip.MainWindow"

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

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

Title="Настройка всплывающей подсказки (ToolTip)" Height="363" Width="398">

<Grid>

<!--

InitialShowDeley – задержка в миллисекундах, указывающая на то, через какой

промежуток времени будет отображена подсказка.

ShowDuration – Сколько подсказка будет отображаться.

BetweenShowDelay – если пользователь в течении времени, которое указано в этом свойстве,

наведет на другой указатель то подсказка появится без замедлений.

ToolTip – Содержимое окна с подсказкой.

HasDropShadow – бросает ли подсказка тень.

ShowOnDisabled – отображать подсказку на отключенных элементах.

Placement, PlacementTarget, PlacementRectangle, HorizontalOffset, VerticalOffset - управление месторасположением подсказки.

-->

<StackPanel HorizontalAlignment="Stretch" Margin="80" Name="stackPanel1" VerticalAlignment="Stretch">

<Button Content="Простой Tooltip"

Height="23"

Name="button1"

ToolTip="Подсказка для кнопки" />

<Button Content="Простой Tooltip и задержка"

Height="23"

Name="button2"

ToolTipService.InitialShowDelay="1000"

ToolTipService.BetweenShowDelay="5000"

ToolTip="Подсказка с InitialShowDelay=1000 и BetweenShowDelay=5000"></Button>

<Button Height="23" Name="button3" Content="Кнопка с расширенной подсказкой">

<Button.ToolTip>

<ToolTip HasDropShadow="False" Background="#AA22CCCC">

<StackPanel>

<TextBlock>Текст подсказки</TextBlock>

<Image Source="/smile.gif"></Image>

</StackPanel>

</ToolTip>

</Button.ToolTip>

</Button>

<Button Content="ToolTip и Offset" Height="23" Name="button4">

<Button.ToolTip>

<ToolTip HorizontalOffset="10" VerticalOffset="20" Content="Offset"></ToolTip>

</Button.ToolTip>

</Button>

<Button Content="Placement" Height="23" Name="button5">

<Button.ToolTip>

<ToolTip Placement="Left" Content="Placement"></ToolTip>

</Button.ToolTip>

</Button>

</StackPanel>

</Grid>

</Window>

## Transperent

<Window x:Class="\_003\_Transparent.Window1"

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

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

Title="Настройка прозрачности в элементах управления"

Height="331"

Width="529">

<Grid>

<Grid.Background>

<ImageBrush ImageSource="background.jpg" Stretch="UniformToFill"/>

</Grid.Background>

<Button Width="100" Height="100" Margin="110,141,298,51"></Button>

<!--Opacity - свойство устанавливающее непрозрачность элемента. 1 - полностью непрозрачный 0 - полностью прозрачный.-->

<Button Name="textBox"

Width="200"

Height="92"

Opacity="0.5"

Margin="175,73,132,127"

Background="Red"

Foreground="White">

</Button>

</Grid>

</Window>

## Focus

<Window x:Class="WPFFocus.MainWindow"

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

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

Title="Focus" Height="231" Width="274">

<Grid>

<StackPanel Margin="5">

<Button Margin="5" Padding="30" Name="cmdFocus">Focused</Button>

<Button Margin="5" Padding="30">Not Focused</Button>

</StackPanel>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

protected override void OnActivated(EventArgs e)

{

base.OnActivated(e);

cmdFocus.Focus(); // Попытка установить фокус элементу управления.

}

}

## MouseMove

<Window x:Class="WpfMouseEvent.MainWindow"

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

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

Title="События мыши" Height="349" Width="395">

<Grid Margin="5">

<Grid.RowDefinitions>

<RowDefinition></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

</Grid.RowDefinitions>

<!--Обработчик события на движение курсора по поверхности элемента-->

<Rectangle Name="rect"

MouseMove="MouseMoved"

Fill="LightBlue" ></Rectangle>

<TextBlock Name="lblInfo" Grid.Row="1"></TextBlock>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void MouseMoved(object sender, MouseEventArgs e)

{

Point pt = e.GetPosition((UIElement)this); // Получение координат мыши, относительно передаваймого элемента.

lblInfo.Text =

String.Format(" ({0}; {1}) в оконных координатах.",

pt.X, pt.Y);

}

}

## MouseCapture

<Window x:Class="MouseCapture.MainWindow"

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

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

Title="Захват мыши" Height="350" Width="525">

<StackPanel>

<Rectangle Fill="LightBlue" Height="150" Name="Rect"

MouseMove="Rectangle\_MouseMove"

MouseDown="Rectangle\_MouseDown"

MouseUp="Rectangle\_MouseUp">

</Rectangle>

<CheckBox Content="Захват мыши" Margin="10" Name="IsCaptureMouse"></CheckBox>

<TextBlock Name="TextBlock1" Margin="10"></TextBlock>

</StackPanel>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void Rectangle\_MouseMove(object sender, MouseEventArgs e)

{

// Проверка состояния левой клавиши мыши.

if (e.LeftButton == MouseButtonState.Pressed)

{

// Получение координат мыши относительно объекта Rect.

Point currentMouse = e.GetPosition(Rect);

TextBlock1.Text = string.Format("x: {0} y:{1}", currentMouse.X, currentMouse.Y);

}

}

private void Rectangle\_MouseDown(object sender, MouseButtonEventArgs e)

{

if (IsCaptureMouse.IsChecked == true)

{

// Захват мыши на объекте Rect.

Rect.CaptureMouse();

}

}

private void Rectangle\_MouseUp(object sender, MouseButtonEventArgs e)

{

if (IsCaptureMouse.IsChecked == true)

{

// Освобождение мыши.

Rect.ReleaseMouseCapture();

}

}

}

## DragAndDrop

<Window x:Class="\_002\_DragAndDrop.Window1"

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

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

Title="Drag &amp; Drop"

Height="300"

Width="300">

<Grid Margin="5">

<Grid.RowDefinitions>

<RowDefinition />

<RowDefinition />

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition />

<ColumnDefinition />

</Grid.ColumnDefinitions>

<TextBox Padding="10"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Text="Перетащите текст из этого поля ввода" />

<!--

В TextBox уже реализовано Drag&Drop текста.

-->

<Label Grid.Row="0"

Grid.Column="1"

Padding="20"

Background="LightGoldenrodYellow"

VerticalAlignment="Center"

HorizontalAlignment="Center"

MouseDown="lblSource\_MouseDown"

Content="Или эту метку" />

<Label Grid.Row="1"

Grid.ColumnSpan="2"

Background="LightGoldenrodYellow"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Padding="20"

Content="На эту метку"

AllowDrop="True"

Drop="lblTarget\_Drop"/>

<!--

AllowDrop="True" - Разрешаем/запрещаем бросать что-либо в этот элемент.

Drop="lblTarget\_Drop" - Событие при броске в элемент чего-либо.

-->

</Grid>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

}

private void lblSource\_MouseDown(object sender, MouseButtonEventArgs e)

{

// sender – объект, на котором произошло данное событие.

Label lbl = sender as Label;

// Создаем источник.

// Копируем содержимое метки Drop.

// 1 параметр: Элемент управления, который будет источником.

// 2 параметр: Данные, которые будут перемещаться.

// 3 параметр: Эффект при переносе.

DragDrop.DoDragDrop(lbl, lbl.Content, DragDropEffects.Copy);

}

// Событие инициируется в момент бросания перетягиваемого элемента на целевой элемент.

private void lblTarget\_Drop(object sender, DragEventArgs e)

{

// Считываем содержимое кэша Drag&Drop и указываем какой тип данных надо считать.

((Label)sender).Content = e.Data.GetData(DataFormats.Text);

}

}

# Window

## Dialog types

<Window x:Class="DialogsTypes.MainWindow"

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

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

Title="Модальные и немодальные окна" Height="270" Width="400">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="200\*" />

<ColumnDefinition Width="200\*" />

</Grid.ColumnDefinitions>

<Button Content="Модальное"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"

Margin="2"

Name="button1"

Grid.Column="0"

Click="button1\_Click" />

<Button Content="Немодальное"

HorizontalAlignment="Stretch"

VerticalAlignment="Stretch"

Margin="2"

Name="button2"

Grid.Column="1"

Click="button2\_Click" />

</Grid>

</Window>

private void button1\_Click(object sender, RoutedEventArgs e)

{

NewWindow window = new NewWindow("Модальное");

window.ShowDialog();

// Код после метода ShowDialog выполнится только тогда, кода диалоговое окно закроется.

}

private void button2\_Click(object sender, RoutedEventArgs e)

{

NewWindow window = new NewWindow("Немодальное");

window.Show();

// Код после метода Show выполнится сразу.

}

}

public partial class NewWindow : Window

{

public NewWindow()

{

InitializeComponent();

}

public NewWindow(string title) //: this()

{

InitializeComponent(); // Переопределяя конструктор мы должны вызывать метод InitializeComponent

Title = title;

}

}

## Window events

<Window x:Class="\_007\_WindowTimeEvent.Window1"

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

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

Title="События окна"

Height="300"

Width="300"

Loaded="Window\_Loaded"

ContentRendered="Window\_ContentRendered"

Activated="Window\_Activated"

Deactivated="Window\_Deactivated"

Closing="Window\_Closing"

Closed="Window\_Closed" >

<Grid>

<ListBox Name="listBox" />

</Grid>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

listBox.Items.Add("InitializeComponent");

}

// Возникает сразу же после первой визуализации окна.

private void Window\_ContentRendered(object sender, EventArgs e)

{

listBox.Items.Add("ContentRendered");

}

// Происходит когда окно полностью инициализировано и готово к взаимодействию.

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

listBox.Items.Add("Loaded");

}

// Возникает, когда пользователь переключается на это окно, а также при первой загрузке окна.

private void Window\_Activated(object sender, EventArgs e)

{

listBox.Items.Add("Activated");

}

// Возникает, когда пользователь переходит на другое окно, а также когда окно закрывается.

private void Window\_Deactivated(object sender, EventArgs e)

{

listBox.Items.Add("Deactivated");

}

// Возникает при закрытии окна. Позволяет отменить операцию закрытия.

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

MessageBoxResult result = MessageBox.Show("Are you shure?", "Confirm", MessageBoxButton.YesNo);

if (result == MessageBoxResult.No)

{

e.Cancel = true;

}

//Window2 w = new Window2();

//if(w.ShowDialog() == false)

// e.Cancel = true; // Отмена закрытия окна.

}

// Возникает после закрытия окна.

private void Window\_Closed(object sender, EventArgs e)

{

MessageBox.Show("Closed");

}

}

<Window x:Class="\_007\_WindowTimeEvent.Window2"

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

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

Title="Window2" Height="300" Width="300">

<Grid>

<Button Content="Yes" HorizontalAlignment="Left" Margin="36,77,0,0" VerticalAlignment="Top" Width="75" Click="Button\_Click\_1"/>

<Button Content="No" HorizontalAlignment="Left" Margin="170,77,0,0" VerticalAlignment="Top" Width="75" Click="Button\_Click\_2"/>

</Grid>

</Window>

public partial class Window2 : Window

{

public Window2()

{

InitializeComponent();

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

DialogResult = true;

}

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

DialogResult = false;

}

}

## Windows position

<Window x:Class="\_005\_WindowPosition.Window1"

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

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

Title="Позиционирование окна"

Height="300"

Width="300"

WindowStartupLocation="CenterScreen">

<Grid>

<Button Name="button1"

Width="100"

Height="22"

Content="Переместить в центр"

Click="button1\_Click" />

</Grid>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

this.WindowStyle = System.Windows.WindowStyle.ToolWindow;

}

// Для Window в XAML коде WindowStartupLocation="CenterScreen" для старта окна в центре экрана.

private void button1\_Click(object sender, RoutedEventArgs e)

{

// Вариант 1

Double width = SystemParameters.FullPrimaryScreenWidth;

Double height = SystemParameters.FullPrimaryScreenHeight;

//// Вариант 2 не учитывается панель задач.

//width = SystemParameters.WorkArea.Width;

//height = SystemParameters.WorkArea.Height;

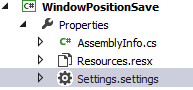
this.Top = (height - this.Height) / 2;

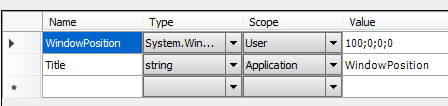
this.Left = (width - this.Width) / 2;

}

}

## Position save





<Window x:Class="\_006\_WindowPositionSave.Window1"

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

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

Title="Сохранение позиции окна"

Height="240"

Width="320">

<Border BorderBrush="White"

BorderThickness="1"

HorizontalAlignment="Center"

VerticalAlignment="Center">

<Image Source="Image.jpg" Stretch="UniformToFill" />

</Border>

</Window>

// Данное приложение для хранения данных использует settings файл.

// Есть два вида настроек settings:

// 1) Application-scoped - эти значения не могут меняться на этапе выполнения. Обычно это

// ConnectionString или подобные значения.

// 2) User-scoped - эти значения могут меняться в процессе выполнения приложения (например, размеры окна, цвет и т.д.)

// Файл settings находится в папке Properties в Solution Explorer

// Данные самого приложения, записанные в процессе выполнения, находятся в папке C:\Documents and Settings\AppData\AppName

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

// Вешаем обработчик на событие перед закрытием окна.

Closing += new CancelEventHandler(Window1\_Closing);

// Восстанавливаем позицию на экране.

Left = Properties.Settings.Default.WindowPosition.Left;

Top = Properties.Settings.Default.WindowPosition.Top;

// Востанавливаем размеры окна.

Width = Properties.Settings.Default.WindowPosition.Width;

Height = Properties.Settings.Default.WindowPosition.Height;

// Востанавливаем заголовок окна.

Title = Properties.Settings.Default.Title;

}

private void Window1\_Closing(object sender, CancelEventArgs e)

{

// RestoreBounds - Возвращает размер и расположение окна перед тем как оно было свернуто или развернуто.

Properties.Settings.Default.WindowPosition = this.RestoreBounds;

//// ОШИБКА! Настройки Application-scoped нельзя изменить.

//Properties.Settings.Default.Title = Title;

// Сохранение настроек.

Properties.Settings.Default.Save();

}

}

nternal sealed partial class Settings {

public Settings() {

// To add event handlers for saving and changing settings, uncomment the lines below:

this.SettingChanging += this.SettingChangingEventHandler;

this.SettingsSaving += this.SettingsSavingEventHandler;

}

private void SettingChangingEventHandler(object sender, System.Configuration.SettingChangingEventArgs e) {

// Add code to handle the SettingChangingEvent event here.

}

private void SettingsSavingEventHandler(object sender, System.ComponentModel.CancelEventArgs e) {

// Add code to handle the SettingsSaving event here.

}

}

## Window interacting

interface IInteractiveWindow

{

void UpdateWindow(string message);

}

<Window x:Class="WindowInteractin2.MainWindow"

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

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

Title="MainWindow" Height="150" Width="451">

<Grid>

<Button Content="Update"

Height="23"

Margin="144,52,151,0"

Name="button1"

VerticalAlignment="Top"

Click="button1\_Click" />

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

// Стартуем окна как немодальные.

Window1 w1 = new Window1();

w1.Show();

Window2 w2 = new Window2();

w2.Show();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

// Перебираем все окна текущего приложения.

for (int i = 0; i < Application.Current.Windows.Count; ++i)

{

Window temp = Application.Current.Windows[i];

// Если окно производное от интерфейса IInteractiveWindow вызываем метод UpdateWindow().

if (temp is IInteractiveWindow)

{

(temp as IInteractiveWindow).UpdateWindow("Hello world");

}

}

}

}

## Window interacting

<Window x:Class="WindowsInteracting.MainWindow"

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

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

Title="Главное окно" Height="175" Width="461">

<Grid>

<Button Content="Показать второе окно"

Height="23"

HorizontalAlignment="Left"

Margin="112,33,0,0"

Name="buttonShow"

VerticalAlignment="Top"

Width="227"

Click="button1\_Click" />

<Button Content="Обновить второе окно"

Height="23"

HorizontalAlignment="Left"

Margin="112,0,0,37"

Name="buttonUpdate"

VerticalAlignment="Bottom"

Width="227"

IsEnabled="False"

Click="buttonUpdate\_Click" />

</Grid>

</Window>

NewWindow \_window = new NewWindow();

public MainWindow()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

// Отображаем второе окно как немодальное.

\_window.Show();

// Делаем первую кнопку не активной.

buttonShow.IsEnabled = false;

// Вторую кнопку, для обновления дочернего окна, делаем активной.

buttonUpdate.IsEnabled = true;

}

private void buttonUpdate\_Click(object sender, RoutedEventArgs e)

{

// Вызываем пользовательский метод, который обновляет значения Label в дочернем окне.

\_window.UpdateWindow("Hello world");

buttonUpdate.IsEnabled = false;

}

}

<Window x:Class="WindowsInteracting.NewWindow"

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

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

Title="NewWindow" Height="300" Width="300">

<Grid>

<Label Name="LabelResult" HorizontalAlignment="Center" VerticalAlignment="Center"> </Label>

</Grid>

</Window>

public partial class NewWindow : Window

{

public NewWindow()

{

InitializeComponent();

}

public void UpdateWindow(string message)

{

LabelResult.Content = message;

}

}

Window ownership

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

// Окна, имеющие окно-владельца, удобно применять для всплывающих окон и панелей инструментов.

// Дочернее окно всегда отображается поверх своего владельца и сворачивается, когда сворачивается окно-владелец.

Window1 window = new Window1();

window.Owner = this;

window.WindowStartupLocation = WindowStartupLocation.CenterOwner;

window.Show();

}

}

<Window x:Class="WindowsOwnership.Window1"

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

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

Title="Window1" Height="100" Width="300">

<Grid>

<Button Click="Button\_Click\_1" Margin="20">Button</Button>

</Grid>

</Window>

public partial class Window1 : Window

{

public Window1()

{

InitializeComponent();

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

(Owner as MainWindow).Title = "Hello from child"; // получение ссылки на родительское окно

}

}

## OpenSave dialogs

<Window x:Class="SaveOpenFileDialogs.MainWindow"

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

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

Title="Главное окно" Height="350" Width="525">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="254\*" />

<ColumnDefinition Width="249\*" />

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="277\*" />

<RowDefinition Height="34" />

</Grid.RowDefinitions>

<TextBox HorizontalAlignment="Stretch"

Grid.ColumnSpan="2"

Margin="10,10,10,4"

Name="textBoxContent"

VerticalAlignment="Stretch"

VerticalScrollBarVisibility="Auto"

/>

<Button Content="Открыть"

Grid.Row="1" Grid.Column="0"

Height="23"

Name="buttonOpen"

HorizontalAlignment="Stretch"

Margin="10,5,5,5"

Click="buttonOpen\_Click" />

<Button Content="Сохранить"

Grid.Row="1" Grid.Column="1"

Height="23"

Name="buttonSave"

HorizontalAlignment="Stretch"

Margin="5,5,10,5"

Click="buttonSave\_Click" />

</Grid>

</Window>

// Из пространства имен Microsoft.Win32

OpenFileDialog \_openDialog = new OpenFileDialog();

SaveFileDialog \_saveDialog = new SaveFileDialog();

public MainWindow()

{

InitializeComponent();

}

private void buttonSave\_Click(object sender, RoutedEventArgs e)

{

// Настраиваем диалоговое окно для сохранения файлов. Указываем два фильтра для расширений файлов.

\_saveDialog.Filter = "Text files (\*.TXT)|\*.txt|All Files (\*.\*)|\*.\*";

if (\_saveDialog.ShowDialog() == true)

{

StreamWriter writer = new StreamWriter(\_saveDialog.FileName);

writer.WriteLine(textBoxContent.Text);

writer.Close();

}

}

private void buttonOpen\_Click(object sender, RoutedEventArgs e)

{

\_openDialog.Filter = "Text files (\*.TXT)|\*.txt|All Files (\*.\*)|\*.\*";

if (\_openDialog.ShowDialog() == true)

{

StreamReader reader = new StreamReader(\_openDialog.FileName);

textBoxContent.Text = reader.ReadToEnd();

reader.Close();

}

}

}

## Arbitrary form

<Window x:Class="NotRectangularWindow.MainWindow"

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

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

Title="Окно не правильной формы" Height="350" Width="525"

AllowsTransparency="True"

WindowStyle="None"

MouseLeftButtonDown="Window\_MouseLeftButtonDown">

<Window.Background>

<ImageBrush ImageSource="bg\_image.png"></ImageBrush>

</Window.Background>

<Grid>

<Button Content="Close"

Height="23"

HorizontalAlignment="Left"

Margin="162,60,0,0"

Name="button1"

VerticalAlignment="Top"

Width="75"

Click="button1\_Click" />

</Grid>

</Window>

/\*

\* 1. Задать свойство AllowTransparency = true

\* 2. Установить WindowsStyle = None

\* 3. В качестве фона установить картинку с прозрачными элементами.

\*/

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void button1\_Click(object sender, RoutedEventArgs e)

{

// Закрываем текущее приложение.

Application.Current.Shutdown();

}

private void Window\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

// Для перетаскивания окна за любую область.

DragMove();

}

}

## Arbitrary form

<Window x:Class="NotRectangularWindow2.MainWindow"

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

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

Title="Окно не правильной формы" Height="350" Width="525"

AllowsTransparency="True"

Background="Transparent"

WindowStyle="None"

MouseLeftButtonDown="Window\_MouseLeftButtonDown">

<Border Background="LightBlue"

CornerRadius="0,20,20,20"

BorderThickness="2"

BorderBrush="Black">

<Grid>

<Button Content="X"

Height="25"

Name="buttonClose"

Width="29"

Margin="6,0,455,270"

Click="buttonClose\_Click" />

</Grid>

</Border>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void buttonClose\_Click(object sender, RoutedEventArgs e)

{

Application.Current.Shutdown();

}

private void Window\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

DragMove();

}

}

## Glass effect

<Window x:Class="GlassEffect.MainWindow"

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

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

Title="Glass Effect" Height="300" Width="500" Loaded="Window\_Loaded" Background="Transparent">

<DockPanel>

<Border DockPanel.Dock="Top" Name="TopBar">

<StackPanel Orientation="Horizontal" Margin="10">

<TextBox Height="23" HorizontalAlignment="Left" Name="textBox1" VerticalAlignment="Top" Width="364" />

<Button Content="OK" Height="24" Margin="5,0,0,0" HorizontalAlignment="Left" Name="button1" VerticalAlignment="Top" Width="84" />

</StackPanel>

</Border>

<!--<Border Background="White" BorderBrush="Gray" BorderThickness="1">

<TextBlock Padding="10">Контент окна</TextBlock>

</Border>-->

</DockPanel>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

try

{

// Сделать часть окна прозрачной.

GlassEffectHelper.ExtendGlass(this, -1, -1, -1, -1);

}

catch

{

MessageBox.Show("Error");

}

}

}

class GlassEffectHelper

{

// Для задания Glass эффекта нужно вызвать метод DwmExtendFrameIntoClientArea из Win32 API

// Импортируем функцию из DwmApi.dll

[DllImport("DwmApi.dll")]

public static extern int DwmExtendFrameIntoClientArea(

IntPtr hwnd,

ref Margins pMarInset);

// Структура для задания области, которая будет прозрачной.

[StructLayout(LayoutKind.Sequential)]

public struct Margins

{

public int cxLeftWidth;

public int cxRightWidth;

public int cyTopHeight;

public int cyBottomHeight;

}

// Выравнивание значений структуры Margins под значение системного DPI

// IntPtr windowHandle - указатель на окно с которым работает данный класс.

public static Margins GetDpiAdjustedMargins(IntPtr windowHandle, int left, int right, int top, int bottom)

{

// Получение системного DPI

System.Drawing.Graphics desktop = System.Drawing.Graphics.FromHwnd(windowHandle);

float DesktopDpiX = desktop.DpiX;

float DesktopDpiY = desktop.DpiY;

// Инициализируем структуру.

Margins margins = new Margins();

// По умолчанию DPI рабочего стола 96dpi.

// Корректируем размеры структуры под системные параметры DPI.

margins.cxLeftWidth = Convert.ToInt32(left \* (DesktopDpiX / 96));

margins.cxRightWidth = Convert.ToInt32(right \* (DesktopDpiX / 96));

margins.cyTopHeight = Convert.ToInt32(top \* (DesktopDpiY / 96));

margins.cyBottomHeight = Convert.ToInt32(bottom \* (DesktopDpiY / 96));

return margins;

}

// 1 параметр - окно для которого делается эффект.

// 2-5 параметры - отступы от краев окна к которым будет применена прозрачность.

// (эффект обязательно должен начинается с края окна и не может быть по центру рабочей области)

public static void ExtendGlass(Window win, int left, int right, int top, int bottom)

{

// WindowInteropHelper - контролирует взаимодействие между WPF и Win32

WindowInteropHelper windowInterop = new WindowInteropHelper(win);

// IntPtr windowHandle - дескриптор текущего окна.

IntPtr windowHandle = windowInterop.Handle;

// HwndSource - объект представляет содержимое WPF приложения в Win32 окне.

HwndSource mainWindowSrc = HwndSource.FromHwnd(windowHandle);

// Устанавливаем окну прозраяный фон.

mainWindowSrc.CompositionTarget.BackgroundColor = Colors.Transparent;

// Корректируем значения структуры Margins

Margins margins =

GetDpiAdjustedMargins(windowHandle, left, right, top, bottom);

// Применяем GlassEffect используя Win32 API.

int returnVal = GlassEffectHelper.DwmExtendFrameIntoClientArea(mainWindowSrc.Handle, ref margins);

if (returnVal < 0)

{

throw new NotSupportedException("Operation failed.");

}

}

}

# Commands

## Command binding

<Window x:Class="NewCommand.MainWindow"

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

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

Title="Использование команд" Height="254" Width="496">

<Grid>

<!-- Привязка команды New для кнопки. Если у команды будет отсутствовать привязка, кнопка будет отключенной. -->

<Button Content="Кнопка"

Height="23"

Width="75"

Command="New"

VerticalAlignment="Center"

HorizontalAlignment="Center" />

</Grid>

</Window>

public MainWindow()

{

InitializeComponent();

// Привязка команды. CommandBinding - пространство имен System.Windows.Input

CommandBinding binding = new CommandBinding(ApplicationCommands.New);

binding.Executed += new ExecutedRoutedEventHandler(binding\_Executed);

this.CommandBindings.Add(binding);

// Для того, что бы команда работала, ее нужно привязать к коллекции команд окна.

// Выполнение команды возможно за счет поднимающихся(Bubble) событий.

// При вызове команды на любом из элементов управления, событие поднимется до уровня

// самого верхнего контейнера - окна, в котором и находится обработчик.

// Системные команды привязаны к горячим клавишам. Команда New - <Ctrl+N>

}

void binding\_Executed(object sender, ExecutedRoutedEventArgs e)

{

MessageBox.Show("Команда 'New' была вызвана.");

}

## Xaml command bindings

<Window x:Class="NewCommandXamlBinding.MainWindow"

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

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

Title="Использование команд. Декларативная привязка команды." Height="350" Width="525">

<!--Привязка команды через XAML-->

<Window.CommandBindings>

<CommandBinding Command="New"

Executed="CommandBinding\_Executed"></CommandBinding>

</Window.CommandBindings>

<Grid>

<Button Width="75"

Height="23"

Command="New">Кнопка</Button>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void CommandBinding\_Executed(object sender, ExecutedRoutedEventArgs e)

{

MessageBox.Show("Команда 'New' была вызвана.");

}

}

## Menu new command

<Window x:Class="NewCommandMenu.MainWindow"

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

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

Title="Использование команд" Height="254" Width="496">

<Window.CommandBindings>

<CommandBinding Command="New"

Executed="CommandBinding\_Executed">

</CommandBinding>

</Window.CommandBindings>

<StackPanel>

<Menu>

<MenuItem Header="File">

<!--В случае с командой для пункта меню, можно не указывать текст самого пункта, так как он будет извлечен из команды.-->

<MenuItem Command="New"></MenuItem>

</MenuItem>

</Menu>

<Button Content="Кнопка"

Command="New"

Margin="0, 30, 0, 30"

Width="75"></Button>

</StackPanel>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void CommandBinding\_Executed(object sender, ExecutedRoutedEventArgs e)

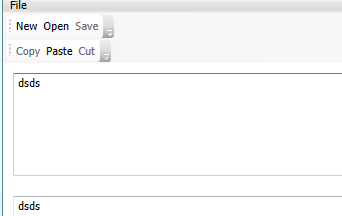
{

MessageBox.Show("Команда 'New' была вызвана.");

}

}

## Simple document



<Window x:Class="SimpleDocument.MainWindow"

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

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

Title="Целевые объекты команд." Height="350" Width="525">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition Height="\*"></RowDefinition>

<RowDefinition Height="\*"></RowDefinition>

</Grid.RowDefinitions>

<Menu Grid.Row="0">

<MenuItem Header="File">

<MenuItem Command="New"></MenuItem>

<MenuItem Command="Open"></MenuItem>

<MenuItem Command="Save"></MenuItem>

<MenuItem Command="SaveAs"></MenuItem>

<Separator></Separator>

<MenuItem Command="Close"></MenuItem>

</MenuItem>

</Menu>

<ToolBarTray Grid.Row="1">

<ToolBar>

<Button Command="New">New</Button>

<Button Command="Open">Open</Button>

<Button Command="Save">Save</Button>

</ToolBar>

<!--

При нажатии на кнопку размещенную в ToolBar, целевым объектом команды будет элемент управления, который находится в фокусе.

TextBox самостоятельно обрабатывает такие команды как Cut, Paste, Copy и т.д. поэтому если TextBox будет в фокусе при

нажатии кнопки в панели задач, TextBox выполнит обработку команды.

Если кнопки с командами находятся за пределами контейнера ToolBar, то им нужно явно указать, с каким контролом они ассоциированы

<Button Command="Cut" CommandTarget="{Binding ElementName=txt}">Cut</Button>

-->

<ToolBar>

<Button Command="Copy">Copy</Button>

<Button Command="Paste">Paste</Button>

<Button Command="Cut">Cut</Button>

</ToolBar>

</ToolBarTray>

<TextBox Name="txt" Grid.Row="2"

TextWrapping="Wrap" AcceptsReturn="True" Margin="10"

TextChanged="txt\_TextChanged"></TextBox>

<TextBox Name="txt2" Grid.Row="3"

TextWrapping="Wrap" AcceptsReturn="True" Margin="10"

TextChanged="txt\_TextChanged"></TextBox>

</Grid>

</Window

public partial class MainWindow : Window

{

// Указывает изменился ли документ с момента открытия.

private bool isDirty = false;

public MainWindow()

{

InitializeComponent();

// Создание привязок.

CommandBinding binding;

binding = new CommandBinding(ApplicationCommands.New);

binding.Executed += NewCommand;

this.CommandBindings.Add(binding);

binding = new CommandBinding(ApplicationCommands.Open);

binding.Executed += OpenCommand;

this.CommandBindings.Add(binding);

binding = new CommandBinding(ApplicationCommands.Save);

binding.Executed += SaveCommand\_Executed;

// Событие, которое вызывается для проверки доступности элемента управления к которому привязана команда.

binding.CanExecute += SaveCommand\_CanExecute;

this.CommandBindings.Add(binding);

}

private void NewCommand(object sender, ExecutedRoutedEventArgs e)

{

MessageBox.Show("Команда New запущена через " + e.Source.ToString());

isDirty = false;

}

private void OpenCommand(object sender, ExecutedRoutedEventArgs e)

{

isDirty = false;

}

private void SaveCommand\_Executed(object sender, ExecutedRoutedEventArgs e)

{

MessageBox.Show("Команда Save запущена через " + e.Source.ToString());

isDirty = false;

}

private void txt\_TextChanged(object sender, RoutedEventArgs e)

{

// Текст в поле ввода был изменен.

isDirty = true;

}

private void SaveCommand\_CanExecute(object sender, CanExecuteRoutedEventArgs e)

{

// Делаем элементы связанные с командой Save активными или не активными.

// WPF самостоятельно решает, когда вызывать этот метод (например, перемещение фокуса, выполнение команды)

// Вызов этого обработчика может происходить достаточно часто, поэтому в нем не желательно использовать код с длительным временем выполнения.

e.CanExecute = isDirty;

// Для того, что бы принудительно вызвать CanExecute для всех команд, нужно вызвать метод CommandManager.InvalidateRequerySuggested();

}

## Custom command

<Window x:Class="CustomCommand.MainWindow"

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

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

xmlns:local="clr-namespace:CustomCommand"

Title="Пользовательские команды" Height="350" Width="525">

<Window.CommandBindings>

<CommandBinding Command="local:DataCommandsLibrary.Requery"

Executed="CommandBinding\_Executed"></CommandBinding>

</Window.CommandBindings>

<Grid>

<Button Command="local:DataCommandsLibrary.Requery"

Margin="100">Повторный зарос (Ctrl + R)</Button>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void CommandBinding\_Executed(object sender, ExecutedRoutedEventArgs e)

{

MessageBox.Show("Произведен повторный запрос.");

}

}

public class DataCommandsLibrary

{

// Пользовательская команда.

private static RoutedUICommand requery;

static DataCommandsLibrary()

{

// InputGestureCollection - горячие клавиши, на которые команда будет реагировать.

InputGestureCollection inputs = new InputGestureCollection();

inputs.Add(new KeyGesture(Key.R, ModifierKeys.Control, "Ctrl+R"));

requery = new RoutedUICommand("Requery", "Requery", typeof(DataCommandsLibrary), inputs);

// 1 параметр: Текст, который будет отображаться, если команда будет присвоена пункту меню.

// 2 параметр: Имя команды.

// 3 параметр: Класс объявляющий команду.

// 4 параметр: Горячие клавиши.

}

public static RoutedUICommand Requery

{

get { return requery; }

}

}

## Custom command

<Window x:Class="\_02\_CustomCommand.MainWindow"

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

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

Title="Пользовательская команда" Height="350" Width="525"

xmlns:local="clr-namespace:\_02\_CustomCommand" Loaded="Window\_Loaded">

<Grid>

<Button Name="Button1" Width="75" Height="24">Кнопка</Button>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

Button1.Command = new CustomCommand();

}

}

public class CustomCommand : ICommand

{

// Событие происходит при изменениях, которые могут повлиять на возможность запуска команды.

public event EventHandler CanExecuteChanged

{

add

{

CommandManager.RequerySuggested += value;

}

remove

{

CommandManager.RequerySuggested -= value;

}

}

// Метод позволяет определить может ли команда быть запущена в своем текущем состоянии.

public bool CanExecute(object parameter)

{

return true;

}

// Метод который сработает при запуске команды.

public void Execute(object parameter)

{

MessageBox.Show("Пользовательская команда сработала.");

}

}

## Custom command

<Window x:Class="\_03\_CustomCommand.MainWindow"

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

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

Title="Пользовательские команды" Height="350" Width="525" Loaded="Window\_Loaded">

<StackPanel>

<TextBox Name="TextBox1" Margin="10"></TextBox>

<Button Name="Button1" Margin="10" CommandParameter="{Binding ElementName=TextBox1, Path=Text}">

Кнопка</Button>

</StackPanel>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

Button1.Command = new CustomCommand(CommandExecute, CanCommandExecute);

}

private void CommandExecute(object parameter)

{

MessageBox.Show("Привет " + Convert.ToString(parameter));

}

private bool CanCommandExecute(object parameter)

{

return TextBox1.Text != string.Empty;

}

}

public class CustomCommand : ICommand

{

readonly Action<object> \_execute;

readonly Predicate<object> \_canExecute;

public CustomCommand(Action<object> execute)

: this(execute, null)

{

}

public CustomCommand(Action<object> execute, Predicate<object> canExecute)

{

if (execute == null)

{

throw new ArgumentNullException("execute");

}

\_execute = execute;

\_canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return \_canExecute == null ? true : \_canExecute.Invoke(parameter);

}

public event EventHandler CanExecuteChanged

{

add

{

CommandManager.RequerySuggested += value;

}

remove

{

CommandManager.RequerySuggested -= value;

}

}

public void Execute(object parameter)

{

\_execute.Invoke(parameter);

}

}

## Command parameters

<Window x:Class="CommandParameters.MainWindow"

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

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

Title="MainWindow" Height="85" Width="523">

<StackPanel>

<!--При вызове команд будут передаватся строковые параметры-->

<Button Command="New" Name="btn" CommandParameter="btn1">Первая кнопка</Button>

<Button Command="New" CommandParameter="btn2">Вторая кнопка</Button>

</StackPanel>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

CommandBinding binding = new CommandBinding(ApplicationCommands.New);

binding.Executed += new ExecutedRoutedEventHandler(binding\_Executed);

this.CommandBindings.Add(binding);

}

void binding\_Executed(object sender, ExecutedRoutedEventArgs e)

{

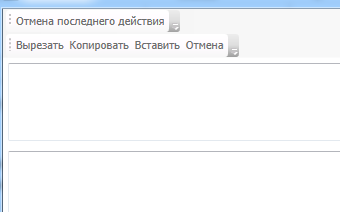
// Чтение параметра команды.

MessageBox.Show(Convert.ToString(e.Parameter));

}

}

## Commands monitor



// Отслеживание и отмена команд.

public partial class MainWindow : Window

{

// Команда Undo, которая будет выполнять отмену в масштабах приложения.

private static RoutedUICommand applicationUndo;

public static RoutedUICommand ApplicationUndo

{

get { return applicationUndo; }

}

static MainWindow()

{

applicationUndo = new RoutedUICommand(

"Application Undo", "ApplicationUndo", typeof(MainWindow));

}

public MainWindow()

{

InitializeComponent();

// Вешаем обработчик для всех событий PreviewExecutedEvent.

// Событие происходит перед тем, как будет вызвана любая команда в приложении.

// На момент срабатывания события, значение задействованного элемента управления еще не изменено.

this.AddHandler(CommandManager.PreviewExecutedEvent,

new ExecutedRoutedEventHandler(PreviewCommandExecute));

}

private void PreviewCommandExecute(object sender, ExecutedRoutedEventArgs e)

{

// Игнорируем, если отправителем является кнопка. Берем только поля ввода.

if (e.Source is ICommandSource) return;

// Игнорируем текущую команду ApplicationUndo

if (e.Command == ApplicationUndo) return;

TextBox txt = e.Source as TextBox;

if (txt != null)

{

RoutedCommand cmd = (RoutedCommand)e.Command;

// создаем запись в журнале истории.

CommandHistoryItem historyItem = new CommandHistoryItem(

cmd.Name, txt, "Text", txt.Text);

ListBoxItem item = new ListBoxItem();

item.Content = historyItem;

lstHistory.Items.Add(item);

}

}

// Выполнение команды отмены

private void ApplicationUndoCommand\_Executed(object sender, RoutedEventArgs e)

{

// Достаем из истории последнюю запись.

CommandHistoryItem historyItem = ((ListBoxItem)lstHistory.Items[lstHistory.Items.Count - 1]).Content as CommandHistoryItem;

// Выполняем отмену.

historyItem.Undo();

// Удаляем запись из истории.

lstHistory.Items.Remove(lstHistory.Items[lstHistory.Items.Count - 1]);

}

private void ApplicationUndoCommand\_CanExecute(object sender, CanExecuteRoutedEventArgs e)

{

// Команда ApplicationUndo активна если журнал истории не пуст.

if (lstHistory == null || lstHistory.Items.Count == 0)

e.CanExecute = false;

else

e.CanExecute = true;

}

}

// Описывает одну запись в истории действий.

public class CommandHistoryItem

{

// Имя выполненной команды.

private string commandName;

public string CommandName

{

get { return commandName; }

set { commandName = value; }

}

// Элемент, для которого была выполнена команда.

private UIElement elementActedOn;

public UIElement ElementActedOn

{

get { return elementActedOn; }

set { elementActedOn = value; }

}

// Свойство, которое было изменено в целевом элементе.

private string propertyActedOn;

public string PropertyActedOn

{

get { return propertyActedOn; }

set { propertyActedOn = value; }

}

// Предыдущего состояния.

private object previousState;

public object PreviousState

{

get { return previousState; }

set { previousState = value; }

}

public CommandHistoryItem(string commandName, UIElement elementActedOn,

string propertyActedOn, object previousState)

{

CommandName = commandName;

ElementActedOn = elementActedOn;

PropertyActedOn = propertyActedOn;

PreviousState = previousState;

}

// Метод для отмены действия.

public void Undo()

{

// Получаем Type элемента, для которого выполнялась команда.

Type elementType = ElementActedOn.GetType();

// Находим свойство по имени.

PropertyInfo property = elementType.GetProperty(PropertyActedOn);

// Устанавливаем свойству сохраненное значение.

property.SetValue(ElementActedOn, PreviousState, null);

}

public override string ToString()

{

return commandName;

}

}

}

<Window x:Class="CommandsMonitor.MainWindow"

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

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

Title="MainWindow" Height="350" Width="525" xmlns:local="clr-namespace:CommandsMonitor">

<Window.CommandBindings>

<CommandBinding Command="local:MainWindow.ApplicationUndo"

Executed="ApplicationUndoCommand\_Executed"

CanExecute="ApplicationUndoCommand\_CanExecute"></CommandBinding>

</Window.CommandBindings>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"></RowDefinition>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<ToolBarTray Grid.Row="0">

<ToolBar>

<Button Command="Cut">Вырезать</Button>

<Button Command="Copy">Копировать</Button>

<Button Command="Paste">Вставить</Button>

<Button Command="Undo">Отмена</Button>

</ToolBar>

<ToolBar>

<Button Command="local:MainWindow.ApplicationUndo">Отмена последнего действия</Button>

</ToolBar>

</ToolBarTray>

<TextBox Margin="5" Grid.Row="1"

TextWrapping="Wrap" AcceptsReturn="True">

</TextBox>

<TextBox Margin="5" Grid.Row="2"

TextWrapping="Wrap" AcceptsReturn="True">

</TextBox>

<ListBox Grid.Row="3" Name="lstHistory" Margin="5" DisplayMemberPath="CommandName"></ListBox>

</Grid>

</Window>

# Resources

## Assembly resources

<Window x:Class="AssemblyResources.Window1"

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

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

Title="Ресурсы сборки" Height="374" Width="312">

<StackPanel>

<Button Click="cmdPlay\_Click" Padding="5">Next</Button>

<Image Name="img" Source="Images/Blue hills.jpg"></Image>

<Button Height="23" Name="buttonGetResource" Click="buttonGetResource\_Click">Извлечь ресурс</Button>

<Button Height="23" Name="buttonGetAllResources" Click="buttonGetAllResources\_Click">Получить все ресурсы</Button>

</StackPanel>

</Window>

public partial class Window1 : System.Windows.Window

{

public Window1()

{

InitializeComponent();

}

private void cmdPlay\_Click(object sender, RoutedEventArgs e)

{

// Использование объекта BitmapImage, который поддерживает работу с ресурсами.

// Относительный путь к ресурсу.

img.Source = new BitmapImage(new Uri("images/winter.jpg", UriKind.Relative));

// Полный путь к ресурсу.

//img.Source = new BitmapImage(new Uri("pack://application:,,,/images/winter.jpg", UriKind.Absolute));

}

// Чтение ресурсов.

private void buttonGetResource\_Click(object sender, RoutedEventArgs e)

{

// Получение нужного ресурса.

StreamResourceInfo resInfo = Application.GetResourceStream(new Uri("test.txt", UriKind.Relative));

MessageBox.Show("Тип ресурса: " + resInfo.ContentType);

// Чтение данных.

StreamReader reader = new StreamReader(resInfo.Stream, System.Text.Encoding.Default);

string textFromRes = reader.ReadToEnd();

reader.Close();

MessageBox.Show(textFromRes);

}

// Получение всех ресурсов.

private void buttonGetAllResources\_Click(object sender, RoutedEventArgs e)

{

Assembly assembly = Assembly.GetAssembly(this.GetType());

string resourceName = assembly.GetName().Name + ".g";

// Позволяет получить доступ к ресурсу (на основе культуры) на этапе выполнения.

ResourceManager rm = new ResourceManager(resourceName, assembly);

using (ResourceSet set = rm.GetResourceSet(CultureInfo.CurrentCulture, true, true))

{

string result = string.Empty;

foreach (DictionaryEntry item in set)

{

result += item.Key.ToString() + Environment.NewLine;

}

MessageBox.Show(result);

}

}

}

## Object resources

<Window x:Class="ObjectResource.MainWindow"

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

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

Title="Работа с ресрусами в XAML разметке" Height="350" Width="525">

<!--Ресурс окна. Определять ресурс нужно всегда перед тем, как ссылаться на него-->

<Window.Resources>

<ImageBrush x:Key="TileBrush" TileMode="Tile"

ViewportUnits="Absolute"

Viewport="0 0 32 32"

ImageSource="happyface.jpg"

Opacity="0.3"></ImageBrush>

</Window.Resources>

<StackPanel Margin="5">

<!--Кнопка, которая использует ресурс окна.-->

<Button Background="{StaticResource TileBrush}"

Padding="5"

FontWeight="Bold"

FontSize="14"

Margin="5">Кнопка с фоном</Button>

<Button Background="{StaticResource TileBrush}" Padding="5"

FontWeight="Bold"

FontSize="14"

Margin="5">Кнопка с фоном</Button>

<!--Кнопка без ресурса-->

<Button Padding="5"

Margin="5"

FontWeight="Bold"

FontSize="14">Обычная кнопка</Button>

<!--Кнопка, которая использует свой ресурс.-->

<Button Padding="5"

Margin="5"

FontWeight="Bold"

FontSize="14" Content="Кнопка с фоном">

<Button.Resources>

<ImageBrush x:Key="TileBrush"

TileMode="Tile"

ViewportUnits="RelativeToBoundingBox"

Viewport="0 0 0.5 0.5"

ImageSource="happyface.jpg"

Opacity="0.3"></ImageBrush>

</Button.Resources>

<!--Применение ресурса в качестве фона кнопки.-->

<Button.Background>

<StaticResource ResourceKey="TileBrush" />

</Button.Background>

</Button>

<Button Background="{StaticResource TileBrush}" Padding="5"

FontWeight="Bold"

FontSize="14"

Margin="5">Кнопка с фоном</Button>

</StackPanel>

</Window>

## Dynamic resources

<Window x:Class="DynamicResources.MainWindow"

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

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

Title="Статические и динамические ресурсы" Height="350" Width="525">

<Window.Resources>

<ImageBrush x:Key="TileBrush"

TileMode="Tile"

ViewportUnits="Absolute"

Viewport="0 0 35 25"

ImageSource="happyface.jpg"

Opacity="0.5"></ImageBrush>

</Window.Resources>

<StackPanel Margin="5">

<!--{DynamicResource TileBrush} - использование динамического ресурса-->

<Button Background="{DynamicResource TileBrush}"

Padding="5"

FontWeight="Bold"

FontSize="14"

Margin="5">Используется динамический ресурс</Button>

<Button Padding="5"

Margin="5"

Click="cmdChange\_Click"

FontWeight="Bold"

FontSize="14">Изменить ресурc</Button>

<!--{StaticResource TileBrush} - использование статического ресурса-->

<Button Background="{StaticResource TileBrush}"

Padding="5"

Margin="5"

FontWeight="Bold"

FontSize="14">Используется статический ресурс</Button>

</StackPanel>

</Window>

## Application resources

<Window x:Class="ApplicationResource.MainWindow"

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

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

Title="Ресурсы Application" Height="350" Width="525">

<StackPanel>

<Button Height="23" Click="Button\_Click">Запустить Window 1</Button>

<Button Height="23" Click="Button\_Click\_1">Запустить Window 2</Button>

<Button Height="23" Click="Button\_Click\_2">Найти ресурс</Button>

</StackPanel>

</Window>

<Application x:Class="ApplicationResource.App"

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

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

StartupUri="MainWindow.xaml">

<Application.Resources>

<ImageBrush x:Key="TileBrush" TileMode="Tile"

ViewportUnits="Absolute" Viewport="0 0 32 32"

ImageSource="happyface.jpg" Opacity="0.3"></ImageBrush>

</Application.Resources>

</Application>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

new Window1().Show();

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

new Window2().Show();

}

private void Button\_Click\_2(object sender, RoutedEventArgs e)

{

// FindResource - метод производит поиск ресурса в текущем элементе,

// а потом в родительских элементах.

// Если ресурс не найден - выбрасывается исключение.

Button btn = sender as Button;

ImageBrush brush = (ImageBrush)btn.FindResource("TileBrush");

MessageBox.Show("Ресурс найден: " + brush.ToString());

// TryFindResource - если ресурс не найден возвращается null

brush = (ImageBrush)btn.TryFindResource("TileBrush\_123");

if (brush == null)

{

MessageBox.Show("Ресурс не найден");

}

}

}

<Window x:Class="ApplicationResource.Window1"

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

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

Title="Window1" Height="300" Width="300">

<Grid>

<Button Background="{StaticResource TileBrush}" Margin="100">Hello world</Button>

</Grid>

</Window>

## System resources

<Window x:Class="SystemResources.MainWindow"

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

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

Title="Системные ресурсы" Height="73" Width="299">

<Grid>

<!--Применение системного значения цвета-->

<Label Foreground="{x:Static SystemColors.WindowTextBrush}">Hello world</Label>

<!--Для того что бы контрол реагировал на изменение настроек системы. Значение должно быть добавлено

как динамический ресурс. Все классы SystemXXX имеют дополнительные свойства, которые возвращают ссылки на ресурсы

Все эти свойства называются так же как и обычные но в конец добавляется слово Key-->

<Label Margin="100,0,0,0"

Foreground="{DynamicResource {x:Static SystemColors.WindowTextBrushKey}}">

Hello world</Label>

<!--Для Label значение задается из code behind-->

<Label Name="Label1"

Margin="200,0,0,0">

Hello world

</Label>

</Grid>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

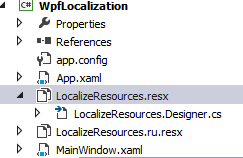
InitializeComponent();

Label1.Foreground = SystemColors.ActiveCaptionBrush;

}

}

## Localization resources



<Window x:Class="WpfLocalization.MainWindow"

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

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

xmlns:p="clr-namespace:WpfLocalization"

Title="Локализация приложения" Height="350" Width="525">

<StackPanel Margin="10">

<TextBlock Text="{x:Static p:LocalizeResources.FNLabel}"></TextBlock>

<TextBox></TextBox>

<TextBlock Text="{x:Static p:LocalizeResources.LNLabel}"></TextBlock>

<TextBox></TextBox>

<TextBlock></TextBlock>

<Button Content="{x:Static p:LocalizeResources.BtnLabel}"></Button>

</StackPanel>

</Window>

## Localization sample

<Window x:Class="WpfLocalizationSample.MainWindow"

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

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

xmlns:l="clr-namespace:LocalizatorHelper;assembly=LocalizatorHelper"

Title="Локализация WPF приложения" Height="350" Width="525">

<Window.Resources>

<l:LocalisationHelper x:Key="helper"></l:LocalisationHelper>

</Window.Resources>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="24"></RowDefinition>

<RowDefinition></RowDefinition>

</Grid.RowDefinitions>

<Menu>

<MenuItem Header="File"></MenuItem>

<MenuItem Header="Language">

<MenuItem Header="Русский" Click="RuMenuItem\_Click"></MenuItem>

<MenuItem Header="English" Click="EnMenuItem\_Click"></MenuItem>

</MenuItem>

</Menu>

<StackPanel Grid.Row="1" Margin="10">

<TextBlock Text="{Binding Source={StaticResource helper}, Path=[MainWindowRes.FN]}"></TextBlock>

<TextBox></TextBox>

<TextBlock Text="{Binding Source={StaticResource helper}, Path=[MainWindowRes.LN]}"></TextBlock>

<TextBox></TextBox>

</StackPanel>

</Grid>

</Window>

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

ResourceManagerService.RegisterManager("MainWindowRes", MainWindowRes.ResourceManager, true);

}

private void RuMenuItem\_Click(object sender, RoutedEventArgs e)

{

ResourceManagerService.ChangeLocale("ru-RU");

}

private void EnMenuItem\_Click(object sender, RoutedEventArgs e)

{

ResourceManagerService.ChangeLocale("en-US");

}

}

public static class ResourceManagerService

{

private static Dictionary<string, ResourceManager> \_managers;

public static event LocaleChangedEventHander LocaleChanged;

private static void RaiseLocaleChanged(CultureInfo newLocale)

{

var evt = LocaleChanged;

if (evt != null)

{

evt.Invoke(null, new LocaleChangedEventArgs(newLocale));

}

}

/// <summary>

/// Текущая культура приложения

/// </summary>

public static CultureInfo CurrentLocale { get; private set; }

static ResourceManagerService()

{

\_managers = new Dictionary<string, ResourceManager>();

ChangeLocale(CultureInfo.CurrentCulture.Name);

}

/// <summary>

/// Получение строкового ресурса по указанному ключу из указанного ResourceManager

///

/// </summary>

/// <param name="managerName">Имя ResourceManager</param>

/// <param name="resourceKey">Имя ресурса для поиска</param>

/// <returns></returns>

public static string GetResourceString(string managerName, string resourceKey)

{

ResourceManager manager = null;

string resource = String.Empty;

if (\_managers.TryGetValue(managerName, out manager))

{

resource = manager.GetString(resourceKey);

}

return resource;

}

/// <summary>

/// Смена текущей культуры

/// </summary>

/// <param name="newLocaleName">Имя культуры (en-US, en-GB)</param>

public static void ChangeLocale(string newLocaleName)

{

CultureInfo newCultureInfo = new CultureInfo(newLocaleName);

Thread.CurrentThread.CurrentCulture = newCultureInfo;

Thread.CurrentThread.CurrentUICulture = newCultureInfo;

CurrentLocale = newCultureInfo;

RaiseLocaleChanged(newCultureInfo);

}

/// <summary>

/// Запускает событие LocaleChanged для обновления привязок

/// </summary>

public static void Refresh()

{

ChangeLocale(CultureInfo.CurrentCulture.IetfLanguageTag);

}

/// <summary>

/// Регистрация ресурс менеджера без обновления интерфейса.

/// </summary>

public static void RegisterManager(string managerName, ResourceManager manager)

{

RegisterManager(managerName, manager, false);

}

/// <summary>

/// Регистрация нового ресурс менеджера и обновление интерфейса.

/// </summary>

public static void RegisterManager(string managerName, ResourceManager manager, bool refresh)

{

ResourceManager \_manager = null;

\_managers.TryGetValue(managerName, out \_manager);

if (\_manager == null)

{

\_managers.Add(managerName, manager);

}

if (refresh)

{

Refresh();

}

}

/// <summary>

/// Удаление ресурс менеджера.

/// </summary>

public static void UnregisterManager(string name)

{

ResourceManager \_manager = null;

\_managers.TryGetValue(name, out \_manager);

if (\_manager != null)

{

\_managers.Remove(name);

}

}

}

# Styles and triggers

## Font resource

<Window x:Class="ReuseFontResource.MainWindow"

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

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

Title="Повторное использование настроек" Height="350" Width="525"

xmlns:sys="clr-namespace:System;assembly=mscorlib">

<!--Ресурсы окна -->

<Window.Resources>

<FontFamily x:Key="ButtonFontFamily">Times New Roman</FontFamily>

<sys:Double x:Key="ButtonFontSize">18</sys:Double>

<FontWeight x:Key="ButtonFontWeight">Bold</FontWeight>

</Window.Resources>

<StackPanel Margin="5">

<Button Padding="5" Margin="5"

FontFamily="{StaticResource ButtonFontFamily}"

FontWeight="{StaticResource ButtonFontWeight}"

FontSize="{StaticResource ButtonFontSize}">

Кнопка с настройками

</Button>

<TextBlock Margin="5">Обычный контент.</TextBlock>

<Button Padding="5" Margin="5">Обычная кнопка</Button>

<TextBlock Margin="5">Еще обычный контент.</TextBlock>

<Button Padding="5" Margin="5"

FontFamily="{StaticResource ButtonFontFamily}"

FontWeight="{StaticResource ButtonFontWeight}"

FontSize="{StaticResource ButtonFontSize}">

Кнопка с настройками

</Button>

</StackPanel>

</Window>

## Font styles

<Window x:Class="ReuseFontWithStyles.MainWindow"

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

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

Title="Использование стиля" Height="350" Width="525">

<Window.Resources>

<!--Объявление стиля c ключом MyButtonStyle-->

<Style x:Key="MyButtonStyle">

<Setter Property="Button.FontFamily" Value="Times New Roman" />

<Setter Property="Button.FontSize" Value="18" />

<Setter Property="Button.FontWeight" Value="Bold" />

</Style>

</Window.Resources>

<StackPanel Margin="5">

<Button Padding="5" Margin="5" FontSize="24"

Style="{StaticResource MyButtonStyle}">

Кнопка со стилем

</Button>

<TextBlock Margin="5">Статический контент</TextBlock>

<Button Padding="5" Margin="5">Обычная кнопка</Button>

<TextBlock Margin="5">Статический контент</TextBlock>

<!--Каждый элемент управления WPF может использовать один стиль.

(Свойство Style определено в базовом классе FrameworlElement)-->

<Button Padding="5" Margin="5"

Style="{StaticResource MyButtonStyle}">

Кнопка со стилем

</Button>

</StackPanel>

</Window>

## Style inheritance

<Window x:Class="StyleInheritance.MainWindow"

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

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

Title="Наследование стилей" Height="350" Width="525">

<Window.Resources>

<Style x:Key="BigFontButtonStyle">

<Setter Property="Control.FontFamily" Value="Times New Roman" />

<Setter Property="Control.FontSize" Value="18" />

<Setter Property="Control.FontWeight" Value="Bold" />

</Style>

<!--Стиль наследует свойства стиля BigFontButtonStyle. Аттрибут BasedOn-->

<Style x:Key="EmphasizedBigFontButtonStyle" BasedOn="{StaticResource BigFontButtonStyle}">

<Setter Property="Control.Foreground" Value="White" />

<Setter Property="Control.Background" Value="DarkBlue" />

</Style>

</Window.Resources>

<StackPanel Margin="5">

<Button Padding="5"

Margin="5"

Style="{StaticResource BigFontButtonStyle}">

Кнопка использует BigFontButtonStyle

</Button>

<TextBlock Margin="5">Статический контент.</TextBlock>

<Button Padding="5" Margin="5">Обычная кнопка</Button>

<TextBlock Margin="5">Статический контент.</TextBlock>

<Button Padding="5"

Margin="5"

Style="{StaticResource EmphasizedBigFontButtonStyle}">

Кнопка использует EmphasizedBigFontButtonStyle

</Button>

</StackPanel>

</Window>

## Button image style

<Window x:Class="StyleComplexValue.MainWindow"

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

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

Title="Сложные значения в настройках стилей" Height="350" Width="525">

<Window.Resources>

<Style x:Key="MyBackgroundButtonStyle">

<Setter Property="Control.Background">

<Setter.Value>

<ImageBrush TileMode="Tile"

Viewport="0 0 0.5 0.5"

ViewportUnits="RelativeToBoundingBox"

ImageSource="happyface.jpg"></ImageBrush>

</Setter.Value>

</Setter>

</Style>

</Window.Resources>

<StackPanel>

<Button Height="100"

Style="{StaticResource MyBackgroundButtonStyle}">

Кнопка со стилем

</Button>

</StackPanel>

</Window>

## Target type

<Window x:Class="StyleProperties.MainWindow"

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

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

Title="Установка стилей определенным типам контролов" Height="350" Width="525">

<Window.Resources>

<!--Стиль будет применяться ко всем объектам типа Button-->

<Style TargetType="Button">

<Setter Property="FontFamily" Value="Times New Roman" />

<Setter Property="FontSize" Value="18" />

<Setter Property="FontWeight" Value="Bold" />

</Style>

</Window.Resources>

<StackPanel Margin="5">

<Button Padding="5" Margin="5">Кнопка со стилем</Button>

<TextBlock Margin="5">Статический контент.</TextBlock>

<!--Для отмены использования стиля Style="{x:Null}-->

<Button Padding="5" Margin="5" Style="{x:Null}">Обчная кнопка</Button>

<TextBlock Margin="5">Статический контент.</TextBlock>

<Button Padding="5" Margin="5">Кнопка со стилем</Button>

</StackPanel>

</Window>

## Event setters

<Window x:Class="EventSetters.MainWindow"

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

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

Title="Использование EventSetter" Height="350" Width="525">

<Window.Resources>

<Style x:Key="MouseOverHighlightStyle">

<Setter Property="TextBlock.Padding" Value="5"/>

<!--Эти обработчики будут срабатывать при событиях MouseEnter и MouseLeave,

на тех контролах, которые используют данный стиль.

Такой подход встречается редко, обычно применяются тригеры.-->

<EventSetter Event="FrameworkElement.MouseEnter"

Handler="element\_MouseEnter" />

<EventSetter Event="FrameworkElement.MouseLeave"

Handler="element\_MouseLeave" />

</Style>

</Window.Resources>

<StackPanel>

<TextBlock Style="{StaticResource MouseOverHighlightStyle}">Наведи на меня.</TextBlock>

<TextBlock Padding="5">Не наводи на меня.</TextBlock>

<TextBlock Style="{StaticResource MouseOverHighlightStyle}">Наведи на меня.</TextBlock>

</StackPanel>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void element\_MouseEnter(object sender, MouseEventArgs e)

{

((TextBlock)sender).Background = new SolidColorBrush(Colors.LightGoldenrodYellow);

}

private void element\_MouseLeave(object sender, MouseEventArgs e)

{

((TextBlock)sender).Background = null;

}

}

## Trigger

<Window x:Class="SimpleTrigger.MainWindow"

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

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

Title="Использование триггеров" Height="350" Width="525">

<Window.Resources>

<Style x:Key="BigFontButtonStyle">

<Style.Setters>

<Setter Property="Control.FontFamily" Value="Times New Roman" />

<Setter Property="Control.FontSize" Value="18" />

</Style.Setters>

<!--Триггеры-->

<Style.Triggers>

<!--Если элемент в фокусе значение Foreground устанавливается как DarkRed-->

<Trigger Property="Control.IsFocused" Value="True">

<Setter Property="Control.Foreground" Value="Yellow" />

</Trigger>

<!--Если курсор находится над элементом, устанавливаются свойства Foreground и FontWeight-->

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

<Setter Property="Control.Foreground" Value="Green" />

<Setter Property="Control.FontWeight" Value="Bold" />

</Trigger>

<!--Если курсор находится над элементом, устанавливаются свойства Foreground и FontWeight-->

<Trigger Property="Button.IsPressed" Value="True">

<Setter Property="Control.Foreground" Value="Red" />

</Trigger>

</Style.Triggers>

</Style>

</Window.Resources>

<StackPanel Margin="5">

<Button Padding="5"

Margin="5"

Style="{StaticResource BigFontButtonStyle}">Кнопка со стилями и триггерами

</Button>

<TextBlock Margin="5">Обычный контент.</TextBlock>

<Button Padding="5"

Margin="5">

Обычная кнопка

</Button>

<TextBlock Margin="5">Обычный контент.</TextBlock>

<Button Padding="5"

Margin="5"

Style="{StaticResource BigFontButtonStyle}">

Кнопка со стилями и триггерами

</Button>

</StackPanel>

</Window>

## Multi trigger

<Window x:Class="MultiTrigger.MainWindow"

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

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

Title="Использование MultiTrigger" Height="350" Width="525">

<Window.Resources>

<Style x:Key="MyButtonStyle">

<Style.Triggers>

<MultiTrigger>

<!--Список условий-->

<MultiTrigger.Conditions>

<Condition Property="Control.IsFocused" Value="True"></Condition>

<Condition Property="Control.IsMouseOver" Value="True"></Condition>

</MultiTrigger.Conditions>

<!--Список изменений, которые вступят в силу, если все условия выполнятся-->

<MultiTrigger.Setters>

<Setter Property="Control.Foreground" Value="DarkBlue"></Setter>

<Setter Property="Control.FontSize" Value="20"></Setter>

</MultiTrigger.Setters>

</MultiTrigger>

</Style.Triggers>

</Style>

</Window.Resources>

<StackPanel>

<Button Name="Button1" Style="{StaticResource MyButtonStyle}" Margin="20">

A Button

</Button>

<Button Name="Button2" Style="{StaticResource MyButtonStyle}" Margin="20">

A Button

</Button>

</StackPanel>

</Window>

## Bindings and triggers

<Window x:Class="DataTrigger.MainWindow"

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

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

xmlns:c="clr-namespace:DataTrigger"

Title="Использование DataTrigger" Height="350" Width="525">

<Window.Resources>

<!--В качестве ресурса используем класс-коллекцию из code behind-->

<c:Countries x:Key="PlacesData"/>

<!--Стиль для ListBoxItem-->

<Style TargetType="ListBoxItem">

<Style.Triggers>

<!--Если значение свойства ShortName объекта с этим стилем будет равно USA - текст станет красного цвета-->

<DataTrigger Binding="{Binding Path=ShortName}" Value="USA">

<Setter Property="Foreground" Value="Red" />

</DataTrigger>

<!--Если значение свойства ShortName и Name будут равны значениям указаным в Value

то для элемента будет пременен стиль указаный в тэге Setter-->

<MultiDataTrigger>

<MultiDataTrigger.Conditions>

<Condition Binding="{Binding Path=Name}" Value="Ukraine" />

<Condition Binding="{Binding Path=ShortName}" Value="UA" />

</MultiDataTrigger.Conditions>

<Setter Property="Background" Value="Cyan" />

</MultiDataTrigger>

</Style.Triggers>

</Style>

</Window.Resources>

<StackPanel>

<TextBlock FontSize="18"

Margin="5"

FontWeight="Bold"

HorizontalAlignment="Center">

Пример работы с DataTrigger

</TextBlock>

<!--ListBox, который использует привязку.-->

<ListBox Width="180"

HorizontalAlignment="Center"

Background="Honeydew"

ItemsSource="{Binding Source={StaticResource PlacesData}}"/>

</StackPanel>

</Window>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

//Resources.Add("PlacesData", new Countries());

}

}

class Countries : ObservableCollection<Country>

{

public Countries()

{

this.Add(new Country("Ukraine", "UA"));

this.Add(new Country("Russia", "RU"));

this.Add(new Country("United States of America", "USA"));

}

}

public class Country

{

public string Name { get; set; }

public string ShortName { get; set; }

public Country()

{

}

public Country(string name, string shortName)

{

Name = name;

ShortName = shortName;

}

public override string ToString()

{

return Name;

}

}

## Trigger events

<Window x:Class="EventTrigger.MainWindow"

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

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

Title="Использование EventTrigger" Height="350" Width="525">

<Window.Resources>

<Style x:Key="BigFontButton">

<!--Стили-->

<Style.Setters>

<Setter Property="Control.FontFamily" Value="Times New Roman" />

<Setter Property="Control.FontSize" Value="18" />

<Setter Property="Control.FontWeight" Value="Bold" />

</Style.Setters>

<!--Триггеры

Обычные тригеры ожидают изменения свойств

EventTrigger - ожидает определенного события-->

<Style.Triggers>

<!--Действие на событие MouseEnter-->

<EventTrigger RoutedEvent="Mouse.MouseEnter">

<EventTrigger.Actions>

<BeginStoryboard>

<Storyboard>

<DoubleAnimation

Duration="0:0:0.2"

Storyboard.TargetProperty="FontSize"

To="22" />

</Storyboard>

</BeginStoryboard>

</EventTrigger.Actions>

</EventTrigger>

<!--Действие на событие MouseLeave-->

<EventTrigger RoutedEvent="Mouse.MouseLeave">

<EventTrigger.Actions>

<BeginStoryboard>

<Storyboard>

<DoubleAnimation

Duration="0:0:1"

Storyboard.TargetProperty="FontSize" To="18" />

</Storyboard>

</BeginStoryboard>

</EventTrigger.Actions>

</EventTrigger>

</Style.Triggers>

</Style>

</Window.Resources>

<StackPanel Margin="5">

<Button Padding="5"

Margin="5"

Style="{StaticResource BigFontButton}">

Кнопка со стилем и триггерами

</Button>

<TextBlock Margin="5">Обычный контент.</TextBlock>

<Button Padding="5" Margin="5">

Обычная кнопка

</Button>

<TextBlock Margin="5">Еще один обычный контент.</TextBlock>

<Button Padding="5"

Margin="5"

Style="{StaticResource BigFontButton}">

Кнопка со стилем и триггерами

</Button>

</StackPanel>

</Window>