# 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"

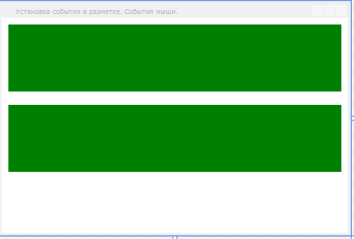
MouseEnter="Rectangle\_MouseEnter" MouseLeave="Rectangle\_MouseLeave"></Rectangle>

<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"

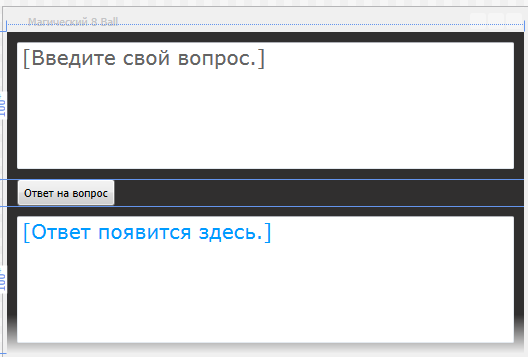
Grid.Row="2"

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"

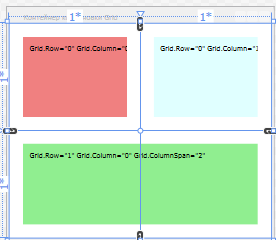
Padding="10" Margin="20" Background="LightGreen">

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>

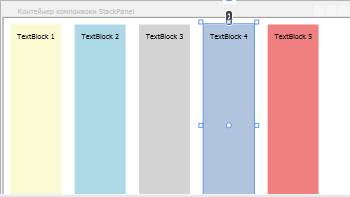
<TextBlock Background="LightGray" Padding="10" Margin="10">TextBlock 3</TextBlock>

<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"

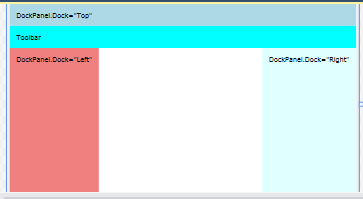
Background="LightCoral"

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>

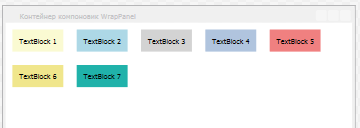
<TextBlock Background="LightCoral" Padding="10" Margin="10">TextBlock 5</TextBlock>

<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"

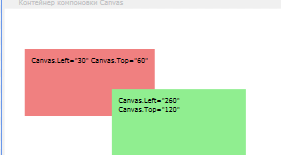
TextWrapping="WrapWithOverflow" Padding="10">

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.

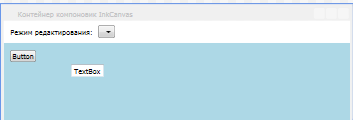
foreach (InkCanvasEditingMode mode in Enum.GetValues(typeof(InkCanvasEditingMode)))

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>

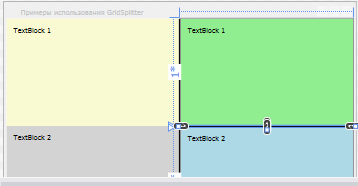
<GridSplitter Grid.Row="1" Grid.Column="0" Height="3"

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

</Grid>

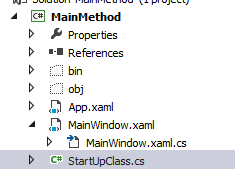
</Grid>

</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");

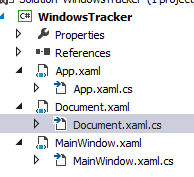
e.Handled = true; // помечаем необработанное исключение, как обработанное.

}

}

}

## 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";

}

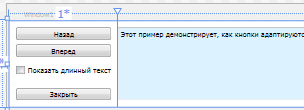
private void cmdClose\_Click(object sender, RoutedEventArgs e)

{

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)

{

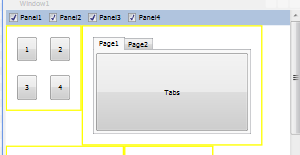
CheckBox chk = e.OriginalSource as CheckBox;

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" />

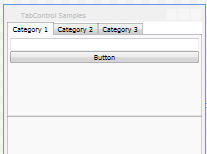
<TabItem Header="Item 2" />

<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 />

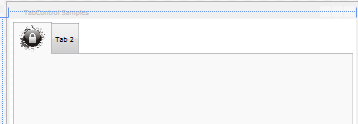
</TabItem>

<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;

}

}

}