# 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"

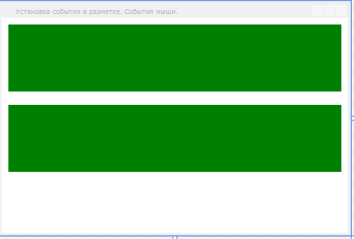
MouseEnter="Rectangle\_MouseEnter" MouseLeave="Rectangle\_MouseLeave"></Rectangle>

<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"

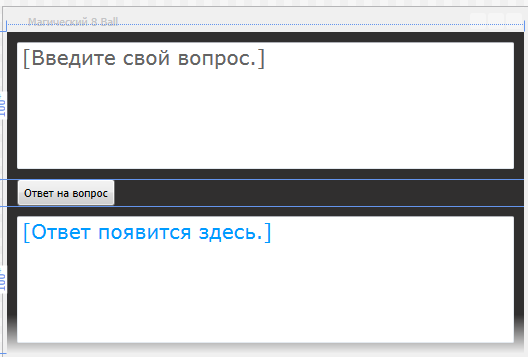
Grid.Row="2"

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"

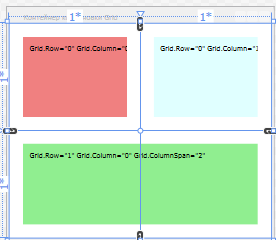
Padding="10" Margin="20" Background="LightGreen">

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>

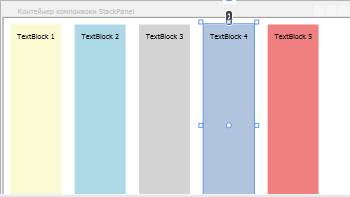
<TextBlock Background="LightGray" Padding="10" Margin="10">TextBlock 3</TextBlock>

<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"

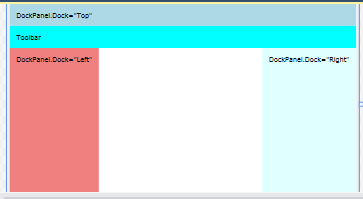
Background="LightCoral"

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>

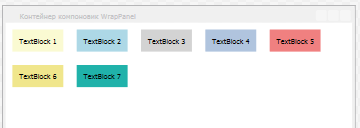
<TextBlock Background="LightCoral" Padding="10" Margin="10">TextBlock 5</TextBlock>

<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"

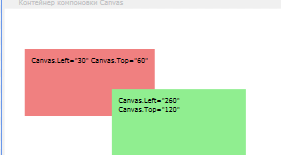
TextWrapping="WrapWithOverflow" Padding="10">

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.

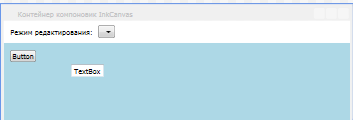
foreach (InkCanvasEditingMode mode in Enum.GetValues(typeof(InkCanvasEditingMode)))

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>

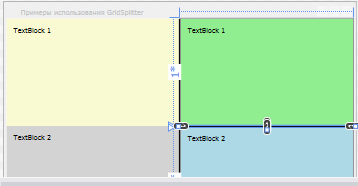
<GridSplitter Grid.Row="1" Grid.Column="0" Height="3"

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

</Grid>

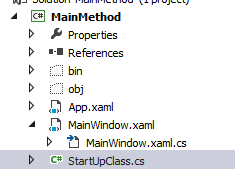
</Grid>

</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");

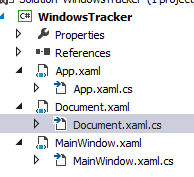
e.Handled = true; // помечаем необработанное исключение, как обработанное.

}

}

}

## 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();

}

}