Урок 5

[Знакомство с технологией WPF. Особенности платформы WPF. Введение в язык XAML. Стили. Обзор элементов управления и их свойств. 1](#_Toc487040090)

[1. Что такое WPF ? 1](#_Toc487040091)

[1.1. WPF vs. WinForms 2](#_Toc487040092)

[1.2 Hello,WPF 2](#_Toc487040093)

[1.2. Работа с App.xaml 5](#_Toc487040094)

[1.3. Запуск приложения с параметрами 6](#_Toc487040095)

[2. XAML 7](#_Toc487040096)

[2.1. Основы XAML 7](#_Toc487040097)

[2.2. Пространства имен XAML 8](#_Toc487040098)

[2.3. События в XAML 9](#_Toc487040099)

[2.4. Регистрация обработчика событий в файле отделенного кода. 11](#_Toc487040100)

[3. Логические ресурсы 11](#_Toc487040101)

[3.1. StaticResource vs. DynamicResource 12](#_Toc487040102)

[3.2. Локальные ресурсы и ресурсы уровня приложения 13](#_Toc487040103)

[3.3. Доступ к ресурсам из файла отделенного кода 14](#_Toc487040104)

[3.4. Словари ресурсов 15](#_Toc487040105)

[4. Стили 17](#_Toc487040106)

[4.1. Стили контролов 18](#_Toc487040107)

[4.2. Стили дочерних контролов 18](#_Toc487040108)

[4.3. Стили, определенные для всего диалогового окна 18](#_Toc487040109)

[4.4. Стили, определенные для всего приложения 19](#_Toc487040110)

[4.5. Явное применение стилей 19](#_Toc487040111)

[5. Обзор элементов управления и их свойств. 20](#_Toc487040112)

[5.1. TextBlock 20](#_Toc487040113)

[5.2. Label 21](#_Toc487040114)

[5.3. TextBox 21](#_Toc487040115)

[5.4. Button 22](#_Toc487040116)

[5.5. CheckBox 22](#_Toc487040117)

[5.6. RadioButton 23](#_Toc487040118)

[5.7. ItemsControl 24](#_Toc487040119)

[5.8. ComboBox 25](#_Toc487040120)

[5.9. ListBox 26](#_Toc487040121)

[6. Window 28](#_Toc487040122)

[7. ДЗ 30](#_Toc487040123)

# Знакомство с технологией WPF. Особенности платформы WPF. Введение в язык XAML. Стили. Обзор элементов управления и их свойств.

1. Что такое WPF ?

Технология WPF (Windows Presentation Foundation) является частью платформы .NET, которая предназначена для построения графических интерфейсов.

WPF представляет собой комбинацию XAML и одного из языков .NET.

XAML (Extensible Application Markup Language — расширяемый язык разметки приложений)

* 1. WPF vs. WinForms

Наиболее заметным отличием WinForms и WPF является тот факт, что WinForms представляет собой всего лишь надстройку над стандартными элементами Windows интерфейса(например, TextBox-ом). WPF же, создан с нуля и не основывается на стандартных элементах Windows, в большинстве случаев.

Наиболее характерным примером отличия технологий WinForms и WPF является Button с изображением и текстом. Т.к. не существует подобного стандартного элемента Windows, то WinForms не предоставляют изначально данный элемент. Вместо этого вы вынуждены разрабатывать собственный Button, который может содержать изображение или использовать графические компоненты сторонних производителей.

В случае с WPF Button может содержать любые элементы, поскольку Button представляет собой рамку с содержимым и различные состояния элемента (нажата, отжата и т.п.). WPF Button не имеет определенного внешнего вида, как и большинство WPF элементов. Это означает, что они могут содержать в себе набор произвольных элементов.

Если вы хотите получить Button с текстом и изображением, то просто добавьте Image и TextBlock внутрь Button.

**Достоинства WPF**

* Новая технология, соответствующая текущим стандартам разработки.
* Широко используется компанией Microsoft, в т.ч. при создании VisualStudio.
* Гибкая технология, т.е. у вас нет необходимости создавать свои контролы или покупать дополнительные контролы.
* XAML облегчает создание и изменение интерфейса вашего приложения, позволяя разделить работу по созданию интерфейсов между дизайнером и программистом.
* Технология связывания данных, позволяет провести четкую границу между данными и разметкой.
* Использование аппаратного ускорения при отображении GUI увеличивает производительность приложения.
* Возможно создание пользовательских интерфейсов для windows- и web-приложений (SilverLight).

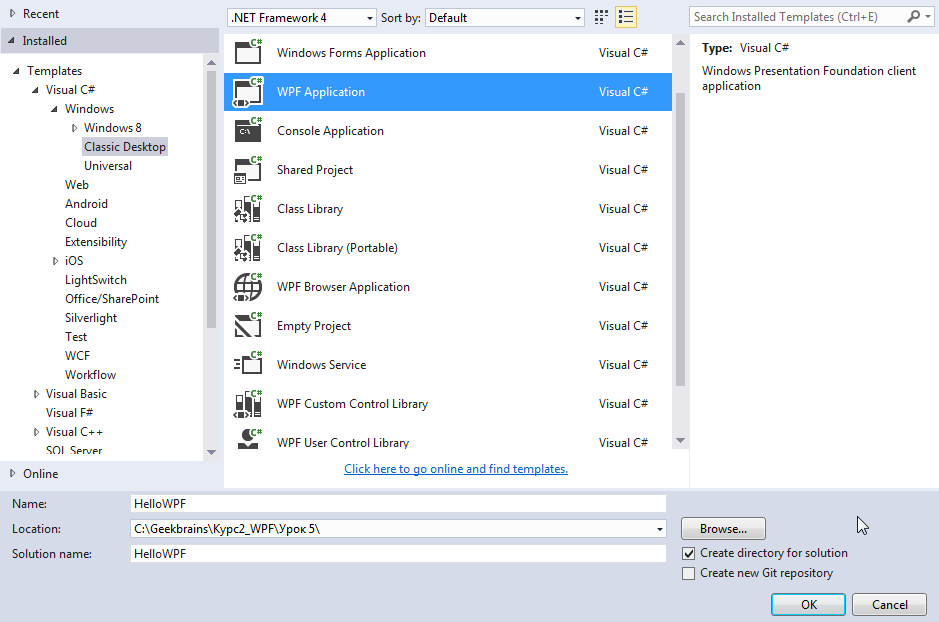
**Достоинства WinForms**

* Накоплен огромный опыт по разработке приложений с использованием WinForms
* Большое количество контролов от сторонних разработчиков, платных и бесплатных.
* Редактор для WinForms удобнее, чем аналогичный редактор для WPF.

# 1.2 Hello,WPF

Классическим первым приложением, при изучении новой технологии или языка программирования, является «Hello, World». Цель такого рода приложений – вывести на экран фрагмент текста и продемонстрировать насколько легко можно начать использовать новую технология или язык программирования.

Создадим с помощью VisualStudio проект HelloWPF, используя шаблон WPF Application.



В результате в папке проекта будет создано несколько файлов, но нас сейчас интересует файл MainWindox.xaml. По-умолчанию, это главная форма нашего приложения. Данная форма отображается на экране после старта приложения.

Код XAML данной формы выглядит следующим образом:

MainWindow.xaml

<Window x:Class="HelloWPF.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:HelloWPF"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid>

</Grid>

</Window>

Добавим контрол TextBox в Grid- панель формы. Установим значение свойства Text у элемента TextBlock равным «Hello, WPF!», размер шрифта и выравнивание текста установите по своему вкусу.

MainWindow.xaml

<Window x:Class="HelloWPF.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:HelloWPF"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid Name="MainGrid">

<TextBlock x:Name="textBlock"

HorizontalAlignment="Center"

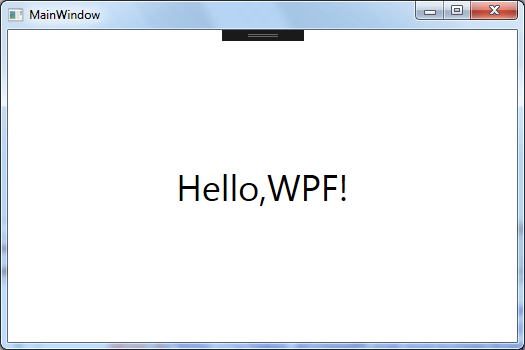
VerticalAlignment="Center"

Text=" Hello, WPF!" FontSize="36"/>

</Grid>

</Window>

Запустите приложение.



* 1. Работа с App.xaml

Жизненный цикл WPF приложение выглядит следующим образом. Объект Application создается вскоре после запуска приложения. Во время его выполнения возникают различные события приложения, которые можно отслеживать. И, наконец, когда объект Application прекращает существовать, приложение завершается.

Файл App.xaml является тем объектом, в котором описывается запуск приложения. VisualStudio создает данный файл каждый раз при создании WPF-приложения, вместе с файлом отделенного кода App.xaml.cs.

Файл App.xaml.cs позволяет расширить класс Application, который является основным классом WPF-приложения. Платформа .Net использует класс Application для определения параметров старта приложения и запуска требуемого Window или Page. Кроме того, класс Application используется для регистрации обработчиков событий, возникающих при старте приложения.

Наиболее часто файл App.xaml используется для описания глобальных ресурсов, использующихся в приложении, например, глобальных стилей.

Пример файла App.xaml

<Application x:Class="HelloWPF.App"

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

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

xmlns:local="clr-namespace:HelloWPF"

StartupUri="MainWindow.xaml">

<Application.Resources>

</Application.Resources>

</Application>

Наибольший интерес в данном файле вызывает свойство StartupUri. C его помощью определяется какое Window или Page будут отображены при старте приложения.

В случаях, когда необходимо контролировать когда и как отображается первое окно, удобнее удалить свойство StartupUri из файла App.xaml и определить процедуру отображения первого окна в файле отделенного кода.

Например, можно зарегистрировать обработчик события Startup, в котором создать первое окно вручную.

App.xaml

<Application x:Class="HelloWPF.App"

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

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

xmlns:local="clr-namespace:HelloWPF"

Startup="Application\_Startup">

<!--StartupUri="MainWindow.xaml"-->

<Application.Resources>

</Application.Resources>

</Application>

App.xaml.cs

using System.Windows;

namespace HelloWPF

{

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

private void Application\_Startup(object sender, StartupEventArgs e)

{

// Создаем первое окно

MainWindow wnd = new MainWindow();

// Определяем необходимые свойства окна

wnd.Title = "Hello, WPF!";

// Отображаем окно

wnd.Show();

}

}

}

* 1. Запуск приложения с параметрами

Для использования параметров при запуске WPF-приложения достаточно в коде обратиться к параметру StartupEventArgs. Он содержит свойство Args, представляющее собой массив строк. Данный массив строк содержит значения параметров приложения.

App.xaml

using System.Windows;

namespace HelloWPF

{

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

private void Application\_Startup(object sender, StartupEventArgs e)

{

// Создаем первое окно

MainWindow wnd = new MainWindow();

if (e.Args.Length == 1)

MessageBox.Show("Параметр: \n\n" + e.Args[0]);

wnd.Show();

}

}

}

1. XAML

XAML представляет собой, основанный на XML, язык разметки для декларативного программирования приложений, разработанный Microsoft. В WinForms графический интерфейс создавался с использованием того же языка программирования, что использовался для взаимодействия с графическим интерфейсом. Т.е. синтаксис С# или VB.Net поддерживался редактором форм (Visual Studio), но в случае с XAML Microsoft пошел другим путем.

Когда вы создаете Window или Page, то они содержат файл XAML и файл отделенного кода (codebehind file). Файл XAML описывает интерфейс со всеми элементами, а файл отделенного кода обрабатывает все события и управляет XAML контролами. При компиляции приложения в Visual Studio код в xaml-файлах также компилируется в бинарное представление кода xaml, которое называется BAML (Binary Application Markup Language). После этого код baml встраивается в финальную сборку приложения - exe или dll-файл.

Теперь, как и в случае с HTML, вы можете легко создавать и изменять графический интерфейс своего приложения.

Возможности XAML не ограничивается только описанием графических интерфейсов: данный язык также используется в технологиях WCF и WorkflowFoundation, где он используется для решения задач никак не связанных с графическим интерфейсом.

* 1. Основы XAML

Для создания контрола XAML достаточно написать название контрола, заключив его в угловые скобки. Теги XAML должны быть «закрытыми», т.е. содержать закрывающий тэг, либо содержать слэш в конце открывающего тэга. Например, <Button></Button> или <Button />.

Большинство контролов позволяют помещать содержимое между открывающим и закрывающим тэгом. Например, Button позволяет указывать отображаемый на кнопке текст следующим образом, <Button> Кнопка </Button>.

В отличие от HTML, XAML чувствителен к регистру, поскольку название контролов должны соответствовать типам в .Net фреймворке. Аналогично, наименования атрибутов контролов чувствительны к регистру. Пример, <Button FontWeight="Bold" Content=" Кнопка" />.

Кроме того существует множество контролов, которые в качестве содержимого принимают не только текст, но и другие контролы.

Пример:

MainWindow.xaml

<Window x:Class="HelloWPF.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:HelloWPF"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid>

<Button FontWeight="Bold" Width="140" Height="30">

<WrapPanel Name="pnlMain">

<TextBlock Foreground="Blue">Разно</TextBlock>

<TextBlock Foreground="Red">цветная</TextBlock>

<TextBlock>кнопка</TextBlock>

</WrapPanel>

</Button>

</Grid>

</Window>

Свойство Content может содержать единственный дочерний элемент, поэтому используется контрол WrapPanel для объединения отдельных контролов TextBlock.

Аналогичная разметка может быть создана, используя только код C#, однако данный код будет проигрывать в наглядности.

Button btn = new Button();

btn.FontWeight = FontWeights.Bold;

WrapPanel pnl = new WrapPanel();

TextBlock txt = new TextBlock();

txt.Text = "Разно";

txt.Foreground = Brushes.Blue;

pnl.Children.Add(txt);

txt = new TextBlock();

txt.Text = "цветная";

txt.Foreground = Brushes.Red;

pnl.Children.Add(txt);

txt = new TextBlock();

txt.Text = "кнопка";

pnl.Children.Add(txt);

btn.Content = pnl;

pnlMain.Children.Add(btn);

* 1. Пространства имен XAML

Наибольшую сложность в понимании пространств имен XAML составляет то, каким образом пространство имен <http://schemas.microsoft.com/winfx/2006/xaml/presentation> в файле XAML соответствует System.Windows.Controls в файле отделенного кода. Данное соответствие, для <http://schemas.microsoft.com/winfx/2006/xaml/presentation> и других пространств имен WPF, жестко зашито внутри сборок WPF вместе с несколькими экземплярами пользовательских XmlnsDefinitionAttribute. XmlnsDefinitionAttribute используется для установления пользовательских соответствий между пространствами имен XAML и .Net.

Пространство имен http://schemas.microsoft.com/winfx/2006/xaml/presentation добавляет в проект следующие пространства имен c#:

System.Windows

System.Windows.Automation

System.Windows.Controls

System.Windows.Controls.Primitives

System.Windows.Data

System.Windows.Documents

System.Windows.Forms.Integration

System.Windows.Ink

System.Windows.Input

System.Windows.Media

System.Windows.Media.Animation

System.Windows.Media.Effects

System.Windows.Media.Imaging

System.Windows.Media.Media3D

System.Windows.Media.TextFormatting

System.Windows.Navigation

System.Windows.Shapes

System.Windows.Shell

Не нужно искать в интернете никаких сайтов с адресом schemas.microsoft.com. Это просто формат записи для пространств имен.

Корневой элемент в файле XAML должен содержать как минимум одно пространство имен для собственного описания и описания дочерних классов. Можно указать дополнительные пространства имен xml, в корневом или дочерних элементах, но каждый из них должен иметь уникальный префикс для его идентификации. Обычно, в файлах WPF XAML, также используется второе пространство имен с префиксом x - xmlns:x=<http://schemas.microsoft.com/winfx/2006/xaml>.

Это пространство имен соответствует System.Windows.Markup в С#, а также определяет некоторые дополнительные директивы для парсера и компилятора XAML.

Пространство имен <http://schemas.openxmlformats.org/markup-compatibility/2006> (обычно с префиксом mc) содержит признак Ignorable, который информирует парсер XAML о том, что нужно игнорировать все элементы(атрибуты), которые не соответствуют типам, существующим в .Net.

Благодаря признаку Ignorable возникает возможность объявлять в ExpressionBlend дополнительные свойства, которые работают в режиме редактирования, но игнорируются при работе приложения. Например:

<StackPanel xmlns="http://schemas.microsoft.com/client/2007"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable="d" d:DesignWidth="100" d:DesignHeight="100">

…

</StackPanel>

mc:Ignorable может содержать список пространств имен, разделенных пробелами.

* 1. События в XAML

WPF, как и большинство современных фреймворков, предназначенных для реализации пользовательского интерфейса, управляется событиями (event drivern).

Все контролы, включая Window, который является потомком Control, предоставляют множество событий, на которые может быть подписано приложение.

Существует огромное кол-во различных типов событий. Большинство контролов содержит события KeyDown, KeyUp, MouseDown, MouseEnter, MouseLeave, MouseUp и т.п.

Язык XAML позволяет конструировать пользовательский интерфейс, но для создания функционирующего приложения необходим способ подключения обработчиков событий.

В обычном случае, если происходит компиляция XAML файла и его объединение с кодом С#, то в первую очередь необходимо в корневом элементе XAML файла объявить класс:

<Window x:Class="HelloWPF.MainWindow"

В файле отделенного кода необходимо объявить класс с тем же названием и наполнить его необходимыми атрибутами и событиями.

using System.Windows;

namespace HelloWPF

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

…

}

}

Этот класс наследуется от класса, именованного элементом XML, в данном случае Window.

Этот «фокус» возможен благодаря средству C#, известному под названием частичные классы (partial class). Частичные классы позволяют разместить реализацию класса в нескольких физических файлах во время разработки и объединить содержимое этих файлов при компиляции в одной сборке. Частичные классы могут применяться во многих сценариях управления кодом, но чаще всего они применяются при автоматической генерации кода.

Рассмотрим пример, в котором реализован обработчик MouseUp в контроле Grid.

В коде XAML необходимо указать имя метода обработчика события MouseUp. Таким образом, мы подписываемся на указанное событие.

MainWindow.xaml.

<Window x:Class="HelloWPF.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:HelloWPF"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid Name="MainGrid" MouseUp="Grid\_MouseUp" Background="LightBlue">

</Grid>

</Window>

Тело метода обработчика события указывается в файле отделенного кода.

MainWindow.xaml.cs

using System.Windows;

namespace HelloWPF

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void Grid\_MouseUp(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

MessageBox.Show("Координаты " + e.GetPosition(this).ToString());

}

}

}

Событие MouseUp использует делегат MouseButtonEventHandler. Данный делегат имеет имеет два параметра. Sender – контрол, вызвавший событие и MouseButtonEventArgs – объект содержащий информацию о координатах курсора мыши.

* 1. Регистрация обработчика событий в файле отделенного кода.

В большинстве случаев удобно использовать, описанный выше, способ регистрации обработчиков событий. Но иногда, может потребоваться зарегистрировать обработчик событий в файле отделенного кода.

MainWindow.xaml.cs

using System.Windows;

namespace HelloWPF

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

MainGrid.MouseUp += MainGrid\_MouseUp;

}

private void MainGrid\_MouseUp(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

MessageBox.Show("Координаты " + e.GetPosition(this).ToString());

}

}

}

1. Логические ресурсы

WPF представляет разработчикам возможность сохранять данные как ресурсы, для отдельного контрола, всего окна или приложения в целом. Здесь речь идет о логических ресурсах, которые могут представлять различные объекты - элементы управления, кисти, коллекции объектов и т.д. Логические ресурсы, сохраненные в свойстве Property .Net классов, доступны для всех дочерних элементов. Базовые классы FrameworkElement и FrameworkContentElemet содержат свойство Resourses (тип System.Windows.ResourceDictionary). поэтому большинство WPF классов содержат данное свойство. Каждый XAML ресурс содержит уникальный ключ, который используется при обращении к ресурсу.

В чем смысл использования ресурсов в WPF?

Они позволяют один раз определить какие-либо ресурсы, которые будут многократно использоваться множеством элементов. Такой подход позволяет исключить повторное объявление ресурсов и экономит память. Кроме того, облегчается поддержка кода – если потребуется изменить определенный ресурс, то будет достаточно внести изменения только в одном месте.

Пример использования ресурсов.

Поскольку в качестве ресурса используется строковая переменная, то необходимо добавить соответствующий namespace в MainWindow.xaml, xmlns:sys="clr-namespace:System;assembly=mscorlib"

Затем необходимо добавить непосредственно сам ресурс, указав значение идентификатора ресурса, x:Key.

<Window.Resources>

<sys:String x:Key="strHelloWPF">Hello, WPF!</sys:String>

</Window.Resources>

После этого, становиться возможным в различных элементах окна MainWindow ссылаться на данный ресурс, используя значение идентификатора ресурса, strHelloWPF.

<Window x:Class="HelloWPF.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:sys="clr-namespace:System;assembly=mscorlib"

xmlns:local="clr-namespace:HelloWPF"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Window.Resources>

<sys:String x:Key="strHelloWPF">Hello, WPF!</sys:String>

</Window.Resources>

<StackPanel Margin="10">

<TextBlock Text="{StaticResource strHelloWPF}" FontSize="56" />

<TextBlock>Еще раз "<TextBlock Text="{StaticResource strHelloWPF}" />", но из ресурсов!</TextBlock>

</StackPanel>

</Window>

* 1. StaticResource vs. DynamicResource

В предыдущем примере использовались статические ресурсы, однако помимо них существует возможность применения динамических ресурсов.

Основное различие между разными типами ресурсов состоит в том, что статические ресурсы устанавливается только один раз, при загрузке XAML файла. А динамические ресурсы могут меняться в течение работы программы. Динамические ресурсы позволяют использовать данные, которые отсутствуют во время разработки приложения и формируются только при выполнении приложения.

Пример одновременного использования статических и динамических ресурсов:

MainWindow.xaml

<Window x:Class="WpfResources.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfResources"

xmlns:sys="clr-namespace:System;assembly=mscorlib"

mc:Ignorable="d"

Height="160"

Width="300"

Title="Ресурсы"

Background="{DynamicResource WindowBackgroundBrush}">

<Window.Resources>

<sys:String x:Key="ComboBoxTitle">Список:</sys:String>

<x:Array x:Key="ComboBoxItems" Type="sys:String">

<sys:String>Элемент #1</sys:String>

<sys:String>Элемент #2</sys:String>

<sys:String>Элемент #3</sys:String>

</x:Array>

<LinearGradientBrush x:Key="WindowBackgroundBrush">

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

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

</LinearGradientBrush>

</Window.Resources>

<StackPanel Margin="10">

<Label Content="{StaticResource ComboBoxTitle}" />

<ComboBox ItemsSource="{StaticResource ComboBoxItems}" />

</StackPanel>

</Window>

В данном примере используются статические ресурсы для определения заголовка и содержимого выпадающего списка, а также динамический ресурс для задания фона диалогового окна.

* 1. Локальные ресурсы и ресурсы уровня приложения

В предыдущих примерах ресурсы определялись на уровне диалогового окна, т.е ресурсы были доступны для каждого элемента данного окна. Если некоторые ресурсы требуются только для конкретного контрола, то возможно добавление ресурсов только для ограниченного множества контролов. Например, вот как можно изменить предыдущий пример, чтобы заголовок выпадающего списка был доступен только для дочерних элементов StackPanel.

MainWindow.xaml

<Window x:Class="WpfResources.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfResources"

xmlns:sys="clr-namespace:System;assembly=mscorlib"

mc:Ignorable="d"

Height="160"

Width="300"

Title="Ресурсы"

Background="{DynamicResource WindowBackgroundBrush}">

<Window.Resources>

<!--<sys:String x:Key="ComboBoxTitle">Список:</sys:String>-->

<x:Array x:Key="ComboBoxItems" Type="sys:String">

<sys:String>Элемент #1</sys:String>

<sys:String>Элемент #2</sys:String>

<sys:String>Элемент #3</sys:String>

</x:Array>

<LinearGradientBrush x:Key="WindowBackgroundBrush">

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

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

</LinearGradientBrush>

</Window.Resources>

<StackPanel Margin="10">

<StackPanel.Resources>

<sys:String x:Key="ComboBoxTitle">Список:</sys:String>

</StackPanel.Resources>

<Label Content="{StaticResource ComboBoxTitle}" />

<ComboBox ItemsSource="{StaticResource ComboBoxItems}" />

</StackPanel>

</Window>

Если есть необходимость доступа к определенным ресурсам из разных окон, то необходимо сохранять ресурсы на уровне приложения, в файле App.xaml.

<Application x:Class="WpfResources.App"

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

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

xmlns:local="clr-namespace:WpfResources"

xmlns:sys="clr-namespace:System;assembly=mscorlib"

StartupUri="MainWindow.xaml">

<Application.Resources>

<sys:String x:Key="ComboBoxTitle">Список:</sys:String>

</Application.Resources>

</Application>

* 1. Доступ к ресурсам из файла отделенного кода

Помимо доступа к ресурсам из файла XAML, возможна работа с ресурсами из файла отделенного кода.

App.xaml

<Application x:Class="WpfResources.App"

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

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

xmlns:local="clr-namespace:WpfResources"

xmlns:sys="clr-namespace:System;assembly=mscorlib"

StartupUri="MainWindow.xaml">

<Application.Resources>

<sys:String x:Key="strApp">Hello, from Application!</sys:String>

</Application.Resources>

</Application>

MainWindow.xaml

<Window x:Class="WpfResources.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfResources"

xmlns:sys="clr-namespace:System;assembly=mscorlib"

mc:Ignorable="d"

Height="160"

Width="300"

Title="Ресурсы">

<Window.Resources>

<sys:String x:Key="strWindow">Hello, from Window!</sys:String>

</Window.Resources>

<DockPanel Margin="10" Name="pnlMain">

<DockPanel.Resources>

<sys:String x:Key="strPanel">Hello, from Panel!</sys:String>

</DockPanel.Resources>

<WrapPanel DockPanel.Dock="Top" HorizontalAlignment="Center" Margin="10">

<Button Name="btnClickMe" Click="btnClickMe\_Click">Click me!</Button>

</WrapPanel>

<ListBox Name="lbResult" />

</DockPanel>

</Window>

MainWindow.xaml.cs

using System.Windows;

namespace WpfResources

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

lbResult.Items.Add(pnlMain.FindResource("strPanel").ToString());

lbResult.Items.Add(this.FindResource("strWindow").ToString());

lbResult.Items.Add(Application.Current.FindResource("strApp").ToString());

}

}

}

Для поиска ресурсов используется метод FindResource(). Поиск ресурсов, в примере выше, производиться сначала в панели, затем на уровне окна и в конце на уровне приложения. Если поиск не обнаруживает запрошенные ресурсы на одном из уровней, тот поиск продолжается на уровне выше, пока не обнаружит искомое значение. Т.е., в принципе, поиск ресурсов в предыдущем примере можно выполнять всегда на уровне панели.

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

{

lbResult.Items.Add(pnlMain.FindResource("strPanel").ToString());

lbResult.Items.Add(pnlMain.FindResource("strWindow").ToString());

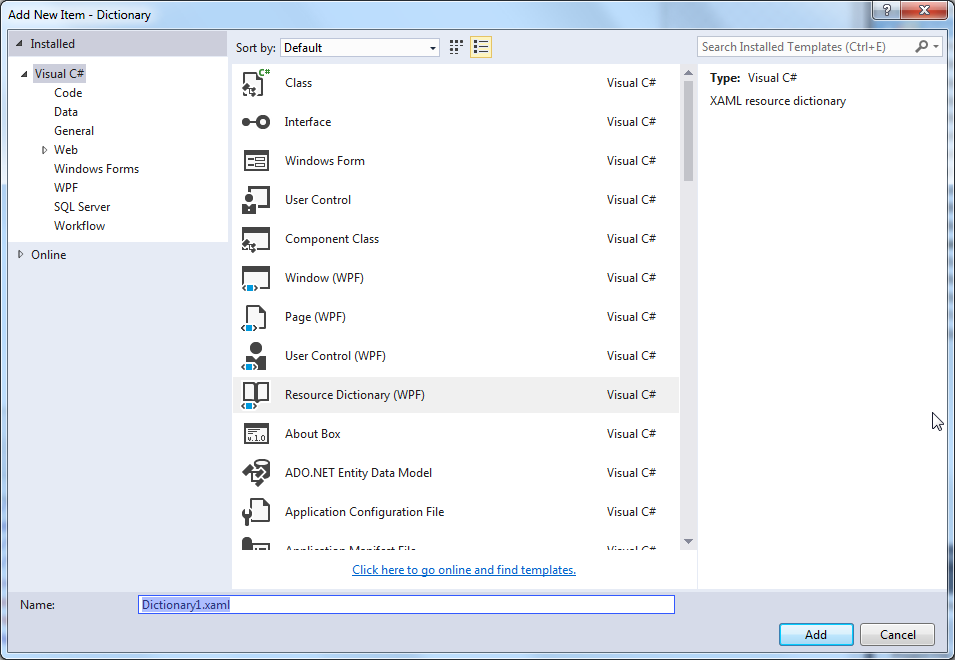
lbResult.Items.Add(pnlMain.FindResource("strApp").ToString());

}

* 1. Словари ресурсов

Существует еще один способ определения ресурсов, который предполагает использование словаря ресурсов.

Добавим в проект словарь с названием Dictionary1.



В файл Dictionary1.xaml добавим градиентную заливку с x:Key="Brush".

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

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

xmlns:local="clr-namespace:Dictionary">

<LinearGradientBrush x:Key="Brush">

<GradientStopCollection>

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

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

</GradientStopCollection>

</LinearGradientBrush>

</ResourceDictionary>

В файл App.xaml добавим информацию о созданном словаре Dictionary1.

Элемент ResourceDictionary.MergedDictionaries представляет собой коллекцию словарей ресурсов, которые добавляются к ресурсам приложения.

<Application x:Class="Dictionary.App"

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

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

xmlns:local="clr-namespace:Dictionary"

StartupUri="MainWindow.xaml">

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<ResourceDictionary Source="Dictionary1.xaml" />

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

</Application>

Загрузку словаря ресурсов необязательно производить через ресурсы приложения. У объекта ResourceDictionary имеется свойство Source, через которое можно связать ресурсы, например, Window, со словарем.

MainWindow.xaml

<Window x:Class="Dictionary.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Dictionary"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Window.Resources>

<ResourceDictionary Source="Dictionary1.xaml" />

</Window.Resources>

<Grid>

<Button Content="Кнопка" MaxHeight="40" MaxWidth="80" Background="{StaticResource Brush}" />

</Grid>

</Window>

1. Стили

Для специалистов знакомых с HTML и CSS, обнаружат, что XAML очень похож на HTML. С помощью тэгов описывается разметка интерфейса приложения. С помощью таких свойств, как Foreground, FontSize и т.п. возможно изменение внешнего вида отдельных контролов.

Аналогично CSS в HTML, в WPF существуют стили, которые позволяют устанавливать внешний вид для группы контролов или для определенного типа контролов. И как в CSS, предусмотрена возможность наследования стилей.

Пример использования стилей:

MainWindow.xaml

<Window x:Class="WpfStyle.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfStyle"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<StackPanel Margin="10">

<StackPanel.Resources>

<Style TargetType="TextBlock">

<Setter Property="Foreground" Value="Black" />

<Setter Property="FontSize" Value="20" />

</Style>

</StackPanel.Resources>

<TextBlock>Заголовок 1</TextBlock>

<TextBlock Foreground="Blue">Заголовок 2</TextBlock>

</StackPanel>

</Window>

В ресурсах контрола StackPanel определен стиль, у которого значение свойства TargetType равно TextBlock. Это означает, что данный стиль применяется ко всем контролам TextBlock внутри StackPanel. В стиль добавлены два Setter-а, с помощью который установлены свойства стилей, ForeGround и FontSize.

Для второго TextBlock свойство ForeGround установлено в Blue. Это демонстрирует возможность изменять значение свойств определенных контролов вне зависимости от используемых стилей.

* 1. Стили контролов

Существует возможность применить стиль исключительно к одному контролу. При этом, стиль определяется, непосредственно, внутри этого контрола.

Например:

MaunWindow.xaml

<Window x:Class="WpfUsingStyle.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfUsingStyle"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid Margin="10">

<TextBlock Text="Style test">

<TextBlock.Style>

<Style>

<Setter Property="TextBlock.FontSize" Value="20" />

</Style>

</TextBlock.Style>

</TextBlock>

</Grid>

* 1. Стили дочерних контролов

Определив стили внутри контрола, становиться возможным применить данные стили для дочерних элементов.

MainWindow.xaml

<Window x:Class="WpfStyle.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfStyle"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<StackPanel Margin="10">

<StackPanel.Resources>

<Style TargetType="TextBlock">

<Setter Property="Foreground" Value="Black" />

<Setter Property="FontSize" Value="20" />

</Style>

</StackPanel.Resources>

<TextBlock>Заголовок 1</TextBlock>

<TextBlock Foreground="Blue">Заголовок 2</TextBlock>

</StackPanel>

</Window>

* 1. Стили, определенные для всего диалогового окна

MainWindow.xaml

<Window x:Class="WpfStyle.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfStyle"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Window.Resources>

<Style TargetType="TextBlock">

<Setter Property="Foreground" Value="Black" />

<Setter Property="FontSize" Value="20" />

</Style>

</Window.Resources>

<StackPanel Margin="10">

<TextBlock>Заголовок 1</TextBlock>

<TextBlock Foreground="Blue">Заголовок 2</TextBlock>

</StackPanel>

</Window>

* 1. Стили, определенные для всего приложения

App.xaml

<Application x:Class="WpfStyle.App"

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

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

xmlns:local="clr-namespace:WpfStyle"

StartupUri="MainWindow.xaml">

<Application.Resources>

<Style TargetType="TextBlock">

<Setter Property="Foreground" Value="Black" />

<Setter Property="FontSize" Value="20" />

</Style>

</Application.Resources>

</Application>

MainWindow.xaml

<Window x:Class="WpfStyle.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfStyle"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<StackPanel Margin="10">

<TextBlock>Заголовок 1</TextBlock>

<TextBlock Foreground="Blue">Заголовок 2</TextBlock>

</StackPanel>

</Window>

* 1. Явное применение стилей

Во всех приведенных выше примерах тот или иной стиль применялся ко всем контролам, тип которых совпадает со значением TargetType в объявлении стиля. Однако, указав для стиля определенное значение свойства x:Key, становиться возможным указать стиль только для конкретного контрола. И это возможно несмотря на указание TargetType. Пример:

MainWindow.xaml

<Window x:Class="WpfStyle.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfStyle"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Window.Resources>

<Style TargetType="TextBlock" x:Key="HeaderStyle">

<Setter Property="Foreground" Value="Black" />

<Setter Property="FontSize" Value="20" />

</Style>

</Window.Resources>

<StackPanel Margin="10">

<TextBlock Style="{StaticResource HeaderStyle}">Заголовок 1</TextBlock>

<TextBlock>Заголовок 2</TextBlock>

</StackPanel>

</Window>

1. Обзор элементов управления и их свойств.
   1. TextBlock

TextBlock позволяет отображать фрагмент текста на экране, почти, также, как и Label, но проделывает это проще и с меньшей затратой ресурсов. Основное отличие TextBlock и Label состоит в следующем:

* Label может отображать одну строку текста, которая может включать другие контролы.
* TextBlock может отображать несколько строк только текста.

TextBlock прекрасно подходит для работы с крупными фрагментами многострочного текста. Но нужно иметь в виду, что если текст не помещается в TextBlock, то WPF отображает только ту часть текста, что поместилась в TextBox.

Чтобы обойти это ограничение существуют следующие приемы:

* LineBreak
* TextTrimming
* TextWrapping

MainWindow.xaml

<Window x:Class="TextBlock.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TextBlock"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<StackPanel>

<TextBlock Margin="10" Foreground="Red">

Меня зовут Бакаре Тунде, я брат первого нигерийского космонавта,<LineBreak />

майора ВВС Нигерии Абака Тунде.

</TextBlock>

<TextBlock Margin="10" TextTrimming="CharacterEllipsis" Foreground="Green">

Мой брат стал первым африканским космонавтом, который отправился с секретной миссией на советскую станцию «Салют-6» в далеком 1979 году.

</TextBlock>

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

Все русские члены команды сумели вернуться на землю, однако моему брату не хватило в корабле места.

</TextBlock>

</StackPanel>

</Window>

* 1. Label

Для Label, в отличие от TextBlock, можно указать набор горячих клавиш, с помощью которых устанавливать фокус на определенный контрол. Набор горячих клавиш формируется из «Alt» и буквы после символа «\_».

MainWindow.xaml

<Window x:Class="Label.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Label"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<StackPanel Margin="10">

<Label Content="\_Name:" Target="{Binding ElementName=txtName}" />

<TextBox Name="txtName" />

<Label Content="\_Mail:" Target="{Binding ElementName=txtMail}" />

<TextBox Name="txtMail" />

</StackPanel>

</Window>

* 1. TextBox

В дополнение к TextBlock и Label, WPF содержит контрол для отображения и редактирования текста, TextBox. Хотя TextBox выглядит довольно просто, он содержит встроенную поддержку для множества функций: Cut, Copy, Paste, Undo, Redo и даже, проверку орфографии. Максимально допустимое количество символов в TextBox можно ограничить с помощью свойства MaxLength. Если необходимо реализовать многострочное текстовое окно для работы с большим объемом текста, то нужно присвоить свойству TextWrapping значение Wrap или WrapWithOverflow.

Wrap не позволяет строкам текста выйти за границы контрола, даже если для этого приходиться разбивать слова. WrapWithOverflow переносит строки, если есть такая возможность, поэтому возможно обрезание текста.

Если свойству AcceptReturn присвоить значение true, то это позволит нажимать кнопку «Enter» для ввода новой строки в TextBox. При этом, программно TextBlock всегда поддерживает многострочный текст. Если свойству Text присвоена строка, содержащая символ новой строки, то будет отображен многострочный текст,независимо от значения свойства AcceptReturn. Также поддержка многострочного текста не зависит от значения свойства TextWrapping.

MainWindow.xaml

<Window x:Class="TextBox.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TextBox"

mc:Ignorable="d"

Title="MainWindow" Height="200" Width="200">

<Grid Margin="10">

<TextBox AcceptsReturn="True" TextWrapping="Wrap" />

</Grid>

</Window>

AcceptReturn разрешает использование перевода строки (Enter) при вводе данных.

* 1. Button

Кнопки – это наиболее знакомый и понятный элемент пользовательского интерфейса.

Хотя большинство представляет, что такое кнопка, точное определение кнопки в WPF не так очевидно. Базовая кнопка является контролом, который поддерживает одинарный клик, но не поддерживает двойной клик. Такое поведение задается абстрактным базовым классом ButtonBase, от которого наследуются все кнопки.

Класс ButtonBase содержит событие Click. Как и в обычных кнопках Windows, это событие формируется после нажатия и отпускания левой клавиши мыши или нажатии кнопки Enter или пробел на клавиатуре, при наличии фокуса на элементе.

Наиболее интересное свойство ButtonBase – это ClickMode. Оно может принимать значение из перечисления ClickMode и таким образом контролировать, когда именно будет сформировано событие Click.

Перечисление ClickMode:

* Release. Значение по умолчанию.Событие Click будет сформировано по нажатия и последующего отпускания кнопки мыши.
* Press. Событие возникнет сразу после нажатия кнопки мыши.
* Hover. После наведения мыши на кнопку и небольшой задержки.

Все кнопки поддерживают горячие клавиши. Для обозначения горячей клавиши служит символ подчеркивания. Когда пользователь нажмет клавишу <Alt> и клавишу доступа, возникнет событие Click данной кнопки.

Класс Button дополняет ButtonBase двумя понятиями – кнопка отмены и кнопка по умолчанию. Если свойству Button.IsCancel присвоено значение true и окно открыто с помощью метода ShowDialog, то окно автоматически будет закрыто с DialogResult равном false. Если свойству Button.IsDefault присвоено значение true, то нажатие кнопки Enter вызовет формирование события Click.

* 1. CheckBox

Элемент CheckBox представляет собой кнопку, которая сохраняет свое состояние после нажатия. Нажатие левой кнопки мыши устанавливает его свойство IsChecked равным true, повторное нажатие меняет значение на false. Контрол содержит свойство IsThreeState. Если значение этого свойства равно true, то свойство IsChecked может принимать три значения: Checked, Unchecked и Intermediate.

В дополнение к свойству IsChecked, контрол содержит отдельные события для каждого значения: Checked для true, Unchecked для false и Indeterminate для null.

MainWindow.xaml

<Window x:Class="CheckBox.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:CheckBox"

mc:Ignorable="d"

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

<StackPanel x:Name="stackPanel">

<CheckBox x:Name="checkBox1" IsThreeState="True" IsChecked="False" Height="20" Content="Неотмечено" />

<CheckBox x:Name="checkBox2" IsThreeState="True" IsChecked="True" Height="20" Content="Отмечено" />

<CheckBox x:Name="checkBox3" IsThreeState="True" IsChecked="{x:Null}" Height="20" Content="Неопределено"/>

<CheckBox x:Name="checkBox" IsChecked="False" Height="20" Content="Флажок"

IsThreeState="True"

Unchecked="checkBox\_Unchecked"

Indeterminate="checkBox\_Indeterminate"

Checked="checkBox\_Checked" />

</StackPanel>

</Window>

MainWindow.xaml.cs

using System.Windows;

namespace CheckBox

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

MessageBox.Show(checkBox.Content.ToString() + " отмечен");

}

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

{

MessageBox.Show(checkBox.Content.ToString() + " не отмечен");

}

private void checkBox\_Indeterminate(object sender, RoutedEventArgs e)

{

MessageBox.Show(checkBox.Content.ToString() + " в неопределенном состоянии");

}

}

}

* 1. RadioButton

Элемент управления RadioButton представляет собой переключатель. Его уникальная особенность - поддержка взаимоисключающих элементов. Если несколько элементов RadioButton объединить в группу, то в один момент времени можно выбрать из этой группы только один элемент. Выбор одного элемента, автоматически снимает признак IsChecked для других элементов.

Для объединения RadioButton в определенную группу, используется свойство GroupName. Все RadioButton с одинаковым значением у свойства GroupName формируют единую группу.

Свойство IsChecked используется для установления значения для RadioButton, а также для определения, в файле отделенного кода, установлен переключатель или нет.

MainWindow.xaml

<Window x:Class="RadioButton.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:RadioButton"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<StackPanel Margin="10">

<Label FontWeight="Bold">Вы знаете С#?</Label>

<RadioButton GroupName="ready">Да</RadioButton>

<RadioButton GroupName="ready">Нет</RadioButton>

<RadioButton GroupName="ready" IsChecked="True">Не уверен</RadioButton>

</StackPanel>

</Window>

* 1. ItemsControl

В WPF все классы для отображения списков являются наследниками абстрактного класса ItemsControl, который в свою очередь является наследником класса Control. ItemsControl хранит свое содержимое в свойстве Items с типом ItemsCollection. Свойство Items является read-only. Это означает, что возможно начальное добавление дочерних элементов в коллекцию, но невозможно присвоить Items другую коллекцию. Для этого у ItemsControl есть другое свойство, ItemsSource, которое поддерживает присвоение ей произвольной коллекции.

MainWindow.xaml

<Window x:Class="ItemsControl.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:system="clr-namespace:System;assembly=mscorlib"

xmlns:local="clr-namespace:ItemsControl"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid Margin="10">

<ItemsControl>

<system:String>Строка #1</system:String>

<system:String>Строка #2</system:String>

<system:String>Строка #3</system:String>

<system:String>Строка #4</system:String>

<system:String>Строка #5</system:String>

</ItemsControl>

</Grid>

</Window>

По умолчанию, ItemsControl не имеет вертикальных и горизонтальных скроллбаров, но существует возможность дополнить ItemsControl данной возможностью. Для этого достаточно поместить элемент ItemsControl внутрь элемента ScrollViewer.

MainWindow.xaml

<Window x:Class="ItemsControl.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:system="clr-namespace:System;assembly=mscorlib"

xmlns:local="clr-namespace:ItemsControl"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid Margin="10">

<ScrollViewer VerticalScrollBarVisibility="Auto" HorizontalScrollBarVisibility="Auto">

<ItemsControl>

<system:String>ItemsControl Item #1</system:String>

<system:String>ItemsControl Item #2</system:String>

<system:String>ItemsControl Item #3</system:String>

<system:String>ItemsControl Item #4</system:String>

<system:String>ItemsControl Item #5</system:String>

</ItemsControl>

</ScrollViewer>

</Grid>

</Window>

* 1. ComboBox

Элемент ComboBox позволяет пользователю выбрать один элемент из списка. Этот элемент достаточно популярен поскольку не занимает много места. Список может быть развернут и свернут по нажатию на кнопке, с помощью комбинации Alt+ стрелка вверх, Alt + стрелка вниз или F4.

У ComboBox определены два события DropDownOpened и DropDownClosed и свойство IsDropDownOpen, которые позволяют выполнять произвольные действия пока ComboBox разворачивает или сворачивает свое содержимое.

ComboBox поддерживает режим в котором пользователь может вводить произвольный текст в ComboBox. Если текст соответствует одному из элементов списка, то данный элемент становиться выбранным. Данный режим управляется с помощью двух свойств: IsEditable и IsReadOnly. По умолчанию, данные свойства имеют значение false.

Свойство StaysOpenOnEdit, при присвоении ему значения true, позволяет сохранить выпадающий список в развернутом виде после нажатия на кнопку.

MainWindow.xaml

<Window x:Class="ComboBoxWPF.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ComboBoxWPF"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid>

<ComboBox x:Name="comboBox" HorizontalAlignment="Center" VerticalAlignment="Center" Width="200" SelectionChanged="comboBox\_SelectionChanged">

<ComboBoxItem>Элемент #1</ComboBoxItem>

<ComboBoxItem>Элемент #2</ComboBoxItem>

<ComboBoxItem>Элемент #3</ComboBoxItem>

</ComboBox>

</Grid>

</Window>

MainWindow.xaml.cs

using System.Windows;

using System.Windows.Controls;

namespace ComboBoxWPF

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void comboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

ComboBox comboBox = (ComboBox)sender;

ComboBoxItem selectedItem = (ComboBoxItem)comboBox.SelectedItem;

MessageBox.Show(selectedItem.Content?.ToString());

}

}

}

* 1. ListBox

Класс ListBox во многом схож с ComboBox, за тем исключением, что все его элементы отображаются в границах контрола или доступны после использования прокрутки.

Класс содержит коллекцию элементов ListBoxItem, которые являются элементами управления содержимым.

Наиболее отличительным свойством ListBox является поддержка множественного выбора элементов. Эта возможность определяется значением свойства SelectionMode, которое может иметь следующие значения:

* Single. Можно выбрать только один элемент, как в ComboBox.
* Multiple. Может быть выбрано произвольное количество элементов. Нажатие левой клавиши мыши на невыбранном элементу приводит к добавлению элемента в коллекцию SelectedItems, а нажатие левой клавиши мыши на выбранном элементе вызывает его удаление из коллекции.
* Extended. Может быть выбрано произвольное количество элементов. Множественный выбор возможен с использованием зажатой клавиши Shift или Cntrl.

Объект ListBox может хранить не только объекты ListBoxItem, но и любые произвольные элементы, т.к. класс ListBoxItem является наследником класса ContentControl, который позволяет хранить фрагменты вложенного содержимого. Если такой фрагмент является классом, порожденным от UIElement, то он будет отображен в элементе ListBox. Если же это другой тип объекта, ListBox вызовет метод ToString() и выведет полученный текст.

Сортировка элементов может быть реализована с использованием объекта ItemsCollection.

ItemsCollection имеет свойство SortDescription, которое может содержать произвольное количество элементов System.ComponentModel.SortDescription. Каждый SortDescription описывает какое свойство элемента используется для сортировки и способ сортировки.

Например:

lbEmployee.Items.SortDescriptions.Add(new SortDescription("Age", ListSortDirection.Ascending));

MainWindow.xaml

<Window x:Class="ListBox.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:ListBox"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid>

<ListBox Margin="10" Name="lbEmployee" SelectionMode="Single"

SelectionChanged="lbEmployee\_SelectionChanged" VerticalAlignment="Top" Height="250"></ListBox>

<Button Margin="10" Height="25" Width="70" VerticalAlignment="Bottom" Click="Button\_Click" >Добавить</Button>

</Grid>

MainWindow.xaml.cs

using System.Collections.ObjectModel;

using System.Windows;

namespace ListBox

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

ObservableCollection<Employee> items = new ObservableCollection<Employee>();

public MainWindow()

{

InitializeComponent();

FillList();

}

void FillList()

{

items.Add(new Employee() { Id = 1, Name = "Vasya", Age = 22, Salary = 3000 });

items.Add(new Employee() { Id = 2, Name = "Petya", Age = 25, Salary = 6000 });

items.Add(new Employee() { Id = 3, Name = "Kolya", Age = 23, Salary = 8000 });

lbEmployee.ItemsSource = items;

lvEmployee.Items.SortDescriptions.Add(new SortDescription("Age", ListSortDirection.Ascending));

}

private void lbEmployee\_Selected(object sender, RoutedEventArgs e)

{

MessageBox.Show(e.Source.ToString());

}

private void lbEmployee\_SelectionChanged(object sender, System.Windows.Controls.SelectionChangedEventArgs e)

{

MessageBox.Show(e.AddedItems[0].ToString());

}

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

{

items.Add(new Employee() { Id = 1, Name = "Sergey", Age = 26, Salary = 7000 });

}

}

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public double Salary { get; set; }

public override string ToString()

{

return $"{Id}\t{Name}\t{Age}\t{Salary}";

}

}

}

1. Window

Для рассмотрения взаимодействия окон, создадим проект WindowRelation. По умолчанию он уже содержит одно главное окно MainWindow.

MainWindow.xaml

<Window x:Class="WindowRelation.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WindowRelation"

mc:Ignorable="d"

Title="MainWindow" Height="350" Width="525">

<Grid>

<Button Width="100" Height="30" Content="Дочернее окно" Click="Button\_Click" />

</Grid>

</Window>

MainWindow.xaml.cs

using System.Windows;

namespace WindowRelation

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

ChildWindow childWindow = new ChildWindow();

childWindow.ViewModel = "ViewModel";

childWindow.Show();

childWindow.ShowViewModel();

}

}

}

Теперь добавим еще одно окно. Для этого в окне добавления нового элемента нам надо выбрать тип "Window (WPF)".

Назовем новое окно ChildWindow. Используя ссылку на окно, мы можем взаимодействовать с ним, например, передавать ему данные из главной формы или вызывать его методы. Например, у окна ChildWindow устанавливается свойство ViewModel и вызывается его метод.

ChildWindow.xaml

<Window x:Class="WindowRelation.ChildWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WindowRelation"

mc:Ignorable="d"

Title="ChildWindow" Height="300" Width="300">

<Grid>

<TextBlock x:Name="textBlock" HorizontalAlignment="Center" TextWrapping="Wrap" Text="TextBlock" VerticalAlignment="Center"/>

</Grid>

</Window>

ChildWindow.xaml.cs

using System.Windows;

namespace WindowRelation

{

/// <summary>

/// Interaction logic for ChildWindow.xaml

/// </summary>

public partial class ChildWindow : Window

{

public ChildWindow()

{

InitializeComponent();

}

public string ViewModel { get; set; }

public void ShowViewModel()

{

textBlock.Text = ViewModel;

}

}

}

Стоит отметить, что после открытия эти окна существуют независимо друг от друга. Можно закрыть главное окно MainWindow, и второе окно ChildWindow все равно продолжит свою работу. Однако, такое поведение несложно изменить.

У всех окон есть свойство Owner, которое указывает на главное окно, владеющее текущим окном. Добавим в обработчик Button\_Click в главном окне следующий текст:

//Теперь MainWindow главное окно для childWindow

childWindow.Owner = this;

Теперь мы можем обращаться из ChildWindow к его владельцу и менять его свойства:

this.Owner.Background = new SolidColorBrush(Colors.Crimson);

Одновременно с этим все дочерние окна доступны в родительском окне через свойство OwnedWindows:

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

{

ChildWindow childWindow = new ChildWindow();

//Теперь MainWindow главное окно для childWindow

childWindow.Owner = this;

childWindow.ViewModel = "ViewModel";

childWindow.Show();

childWindow.ShowViewModel();

foreach (Window window in this.OwnedWindows)

{

window.Background = new SolidColorBrush(Colors.Aquamarine);

if (window is ChildWindow)

window.Title = "Новый заголовок";

}

}

Кроме этого, класс App - главный класс приложения содержит свойство Windows, которое хранит информацию обо всех открытых окнах приложения. И в любом месте программы можно получить эту информацию:

foreach (Window window in App.Current.Windows)

{

window.Background = new SolidColorBrush(Colors.Aquamarine);

// если окно - объект TaskWindow

if (window is ChildWindow)

window.Title = "Новый заголовок!";

}

1. ДЗ

Cоздайте WPF приложение для ведения списка сотрудников компании.  
1. Создайте сущности Employee и Department и заполните списки сущностей начальными данными.  
2. Для списка сотрудников и списка департаментов предусмотрите визуализацию (отображение). Это можно сделать, например, с использованием ComboBox или ListView.  
3. Предусмотрите возможность редактирования сотрудников и департаментов. Должна быть возможность изменить департамент у сотрудника. Список департаментов для выбора, можно выводить в ComboBox, это все можно выводить на дополнительной форме.  
4. Предусмотрите возможность создания новых сотрудников и департаментов. Реализуйте данную возможность либо на форме редактирования, либо сделайте новую форму.