**Практическая работа**

**Технология WPF C#.**

**Цель работы:** Изучить возможности построения графических интерфейсов платформы WPF.

В ходе выполнения практической работы студент должен:

уметь:

* принимать решение о расширении функциональности информационной системы;
* проводить анализ предметной области.

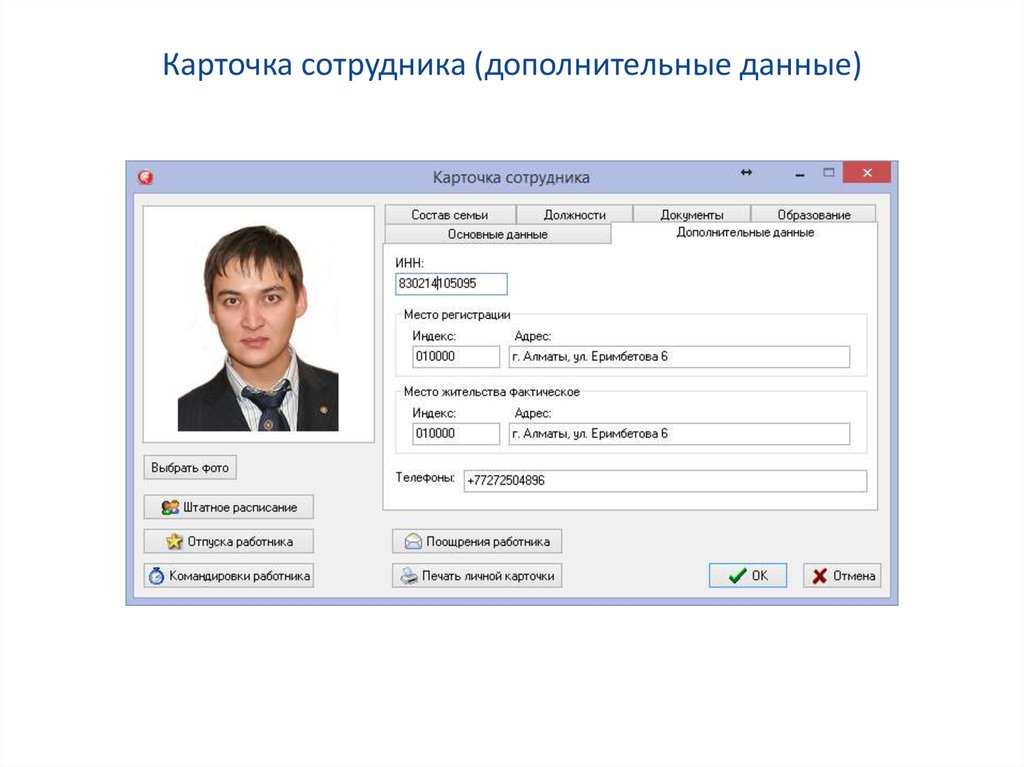
знать:

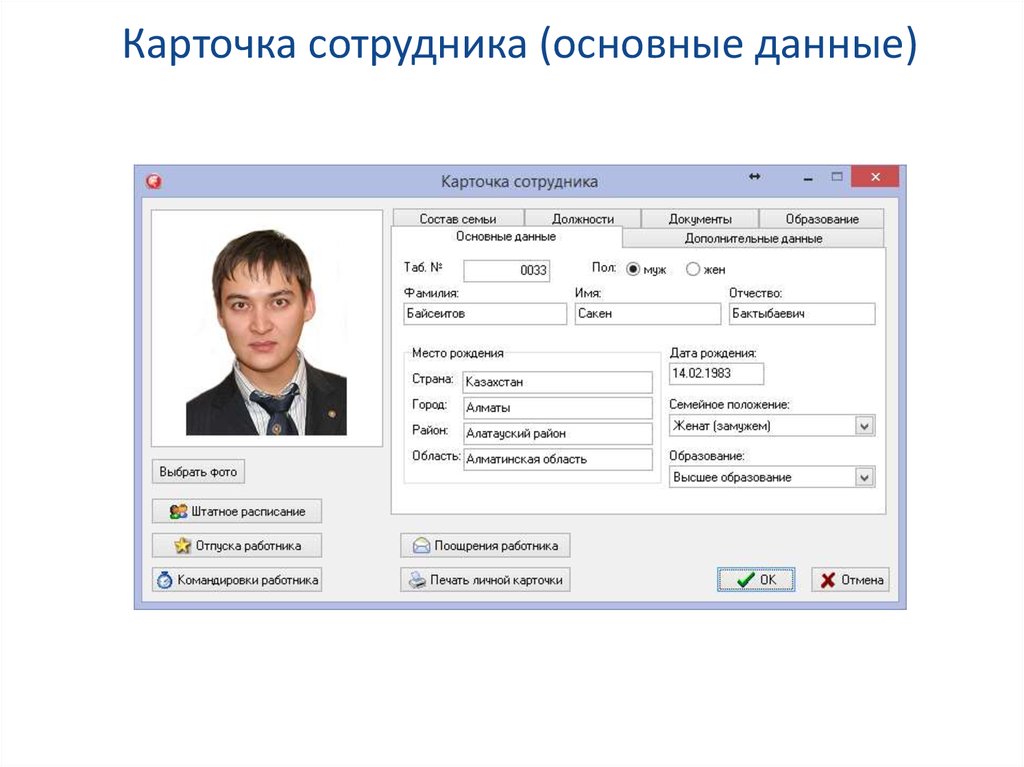
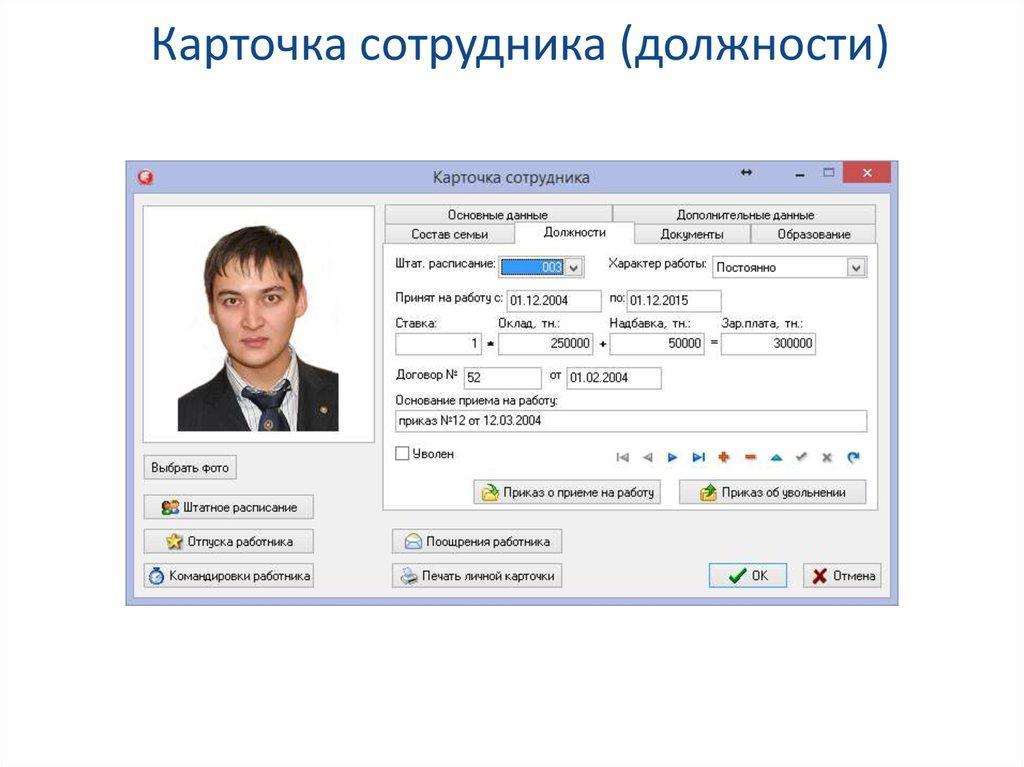
* задачи и функции информационных систем;
* особенности программных средств используемых в разработке информационных систем;

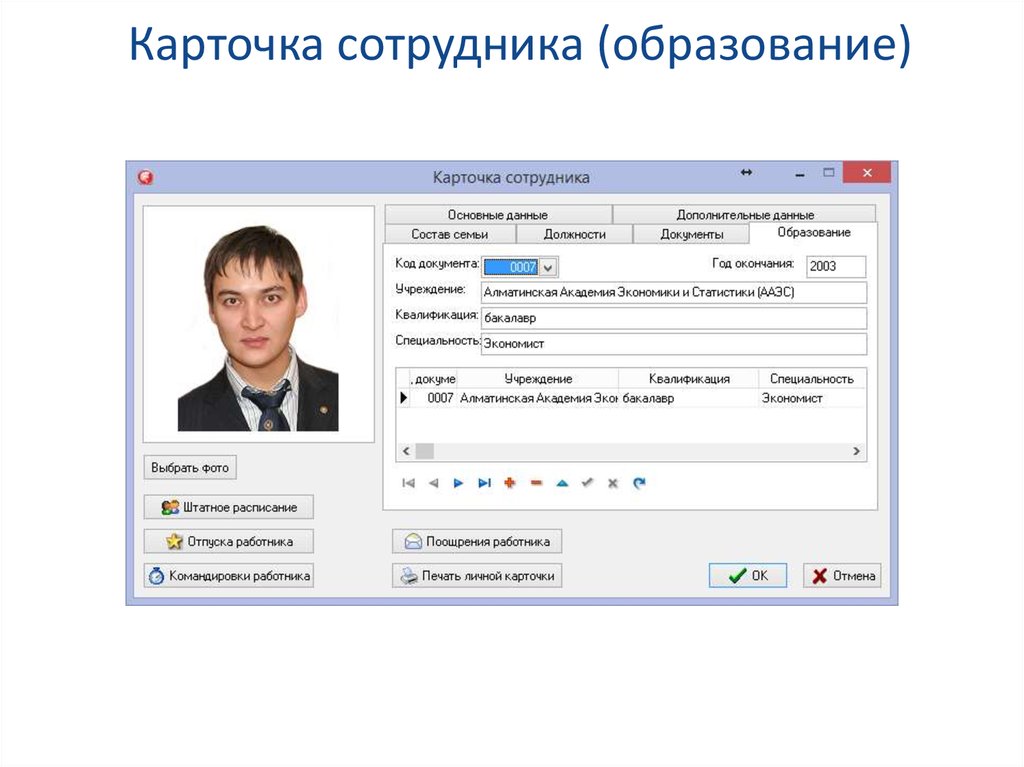
методы и средства проектирования информационных систем.

**Практическая часть**

**Создание графического интерфейса**







**Теоретическая часть**

Технология WPF (Windows Presentation Foundation) является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Если при создании традиционных приложений на основе WinForms за отрисовку элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на **DirectX**. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML: вы можете создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C# и VB.NET, либо совмещать и то, и другое.

**Преимущества WPF**

* Использование традиционных языков .NET-платформы - C# и VB.NET для создания логики приложения
* Возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, основанном на xml и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать XAML и C#/VB.NET
* Независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением.
* Новые возможности, которых сложно было достичь в WinForms, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др.
* Хорошее взаимодействие с WinForms, благодаря чему, например, в приложениях WPF можно использовать традиционные элементы управления из WinForms.
* Богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое
* Аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D, графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной.
* Создание приложений под множество ОС семейства Windows - от Windows XP до Windows 10

В тоже время WPF имеет определенные ограничения. Несмотря на поддержку трехмерной визуализации, для создания приложений с большим количеством трехмерных изображений, прежде всего игр, лучше использовать другие средства - DirectX или специальные фреймворки, такие как Monogame или Unity.

Также стоит учитывать, что по сравнению с приложениями на Windows Forms объем программ на WPF и потребление ими памяти в процессе работы в среднем несколько выше. Но это с лихвой компенсируется более широкими графическими возможностями и провышенной производительностью при отрисовке графики.

**Контейнер Grid**

Это наиболее мощный и часто используемый контейнер, напоминающий обычную таблицу. Он содержит столбцы и строки, количество которых задает разработчик. Для определения строк используется свойство RowDefinitions, а для определения столбцов - свойство ColumnDefinitions.

Каждая строка задается с помощью вложенного элемента RowDefinition, который имеет открывающий и закрывающий тег. При этом задавать дополнительную информацию необязательно.

Каждый столбец задается с помощью вложенного элемента ColumnDefinition. Таким образом в итоге у нас получится таблица 3х3.

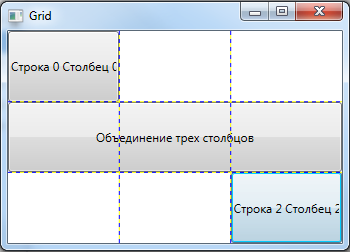
Чтобы задать позицию элемента управления с привязкой к определенной ячейке Grid, в разметке элемента нужно прописать значения свойств Grid.Column и Grid.Row, тем самым указывая, в каком столбце и строке будет находиться элемент. Кроме того, если мы хотим растянуть элемент управления на несколько строк или столбцов, то можно указать свойства Grid.ColumnSpan и Grid.RowSpan

1. Создайте новое приложение LayoutApp как в следующем примере:



Атрибут ShowGridLines="True" у элемента Grid задает видимость сетки, по умолчанию оно равно False.

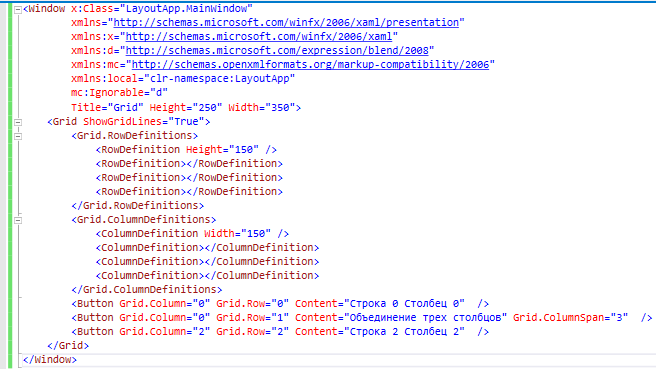
1. Скомпилируйте код



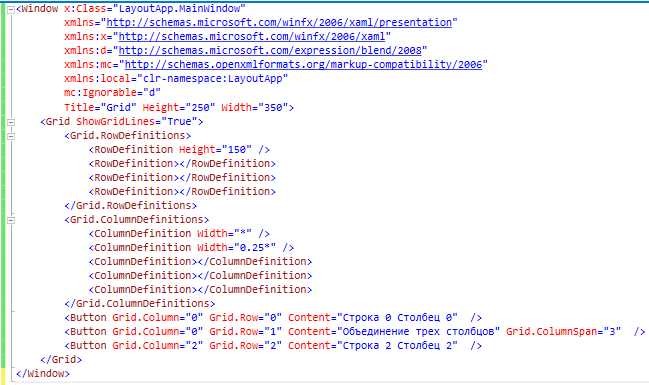
1. Измените код, по разному настроив размеры ячеек. Просмотрите результат
2. Автоматический режим. Здесь столбец или строка занимает то место, которое им нужно.



1. Абсолютный размер. В этом случае высота и ширина указываются в единицах, независимых от устройства:



1. Пропорциональный размер. Например: задаются два столбца, второй из которых имеет ширину в четверть от ширины первого:

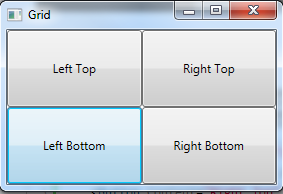


Можно комбинировать все типы размеров. В этом случае от ширины/высоты грида отнимается ширина/высота столбцов/строк с абсолютными или автоматическими размерами, и затем оставшееся место распределяется между столбцами/строками с пропорциональными размерами

**Контейнер UniformGrid**

Аналогичен контейнеру Grid контейнер UniformGrid, только в этом случае все столбцы и строки одинакового размера и используется упрощенный синтаксис для их определения:





**Контейнер GridSplitter**

Элемент GridSplitter помогает создавать интерфейсы наподобие элемента SplitContainer в WinForms, только более функциональные. Он представляет собой некоторый разделитель между столбцами или строками, путем сдвига которого можно регулировать ширину столбцов и высоту строк. В качестве примера можно привести стандартный интерфейс проводника в Windows, где разделительная полоса отделяет древовидный список папок от панели со списком файлов. Например:

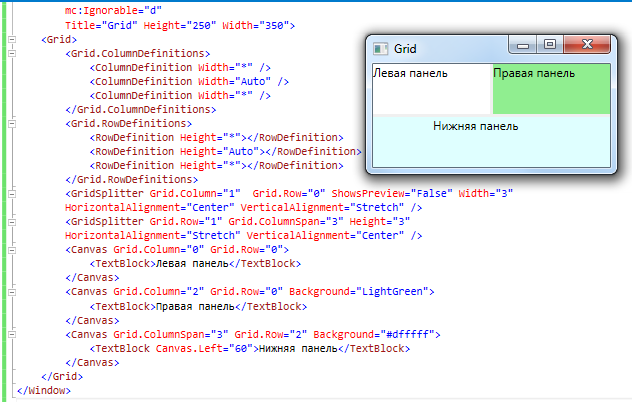


Двигая центральную линию, разделяющую правую и левую части, мы можем устанавливать их ширину.

Итак, чтобы использовать элемент GridSplitter, нам надо поместить его в ячейку в Gride. По сути это обычный элемент, такой же, как кнопка. Как выше, у нас три ячейки (так как три столбца и одна строка), и GridSplitter помещен во вторую ячейку. Обычно строка или столбец, в которые помещают элемент, имеет для свойств Height или Width значение Auto.

Если у нас несколько строк, и мы хотим, чтобы разделитель распространялся на несколько строк, то мы можем задать свойство Grid.RowSpan:

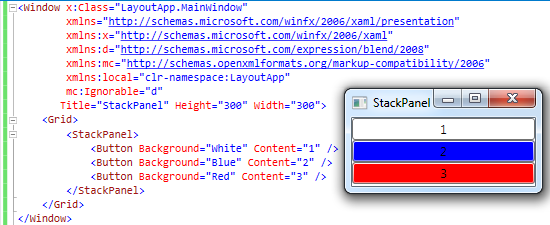
В отличие от элемента SplitContainer в WinForms, в WPF можно установить различное количество динамически регулируемых частей окна.



Здесь у нас сразу два сплитера: один между двумя верхними и нижней панелями, а второй - между правой и левой панелями.

**Контейнер StackPanel**

Это более простой элемент компоновки. Он располагает все элементы в ряд либо по горизонтали, либо по вертикали в зависимости от ориентации. Например,



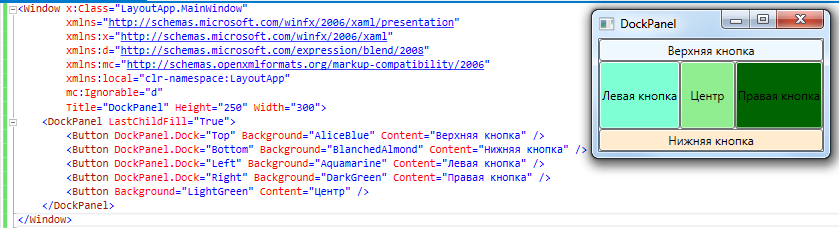
В данном случае для свойства Orientation по умолчанию используется значение Vertical, то есть StackPanel создает вертикальный ряд, в который помещает все вложенные элементы сверху вниз. Мы также можем задать горизонтальный стек. Для этого нам надо указать свойство Orientation="Horizontal":



При горизонтальной ориентации все вложенные элементы располагаются слева направо. Если мы хотим, чтобы наполнение стека начиналось справа налево, то нам надо задать свойство FlowDirection: <StackPanel Orientation="Horizontal" FlowDirection="RightToLeft">. По умолчанию это свойство имеет значение LeftToRight - то есть слева направо.

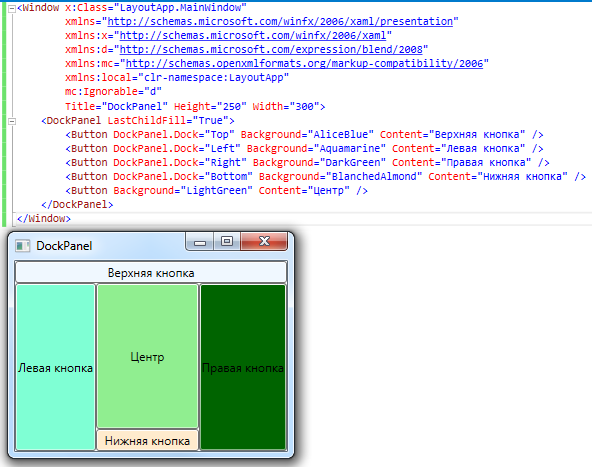
**Контейнер DockPanel**

Этот контейнер прижимает свое содержимое к определенной стороне внешнего контейнера. Для этого у вложенных элементов надо установить сторону, к которой они будут прижиматься с помощью свойства DockPanel.Dock. Например,

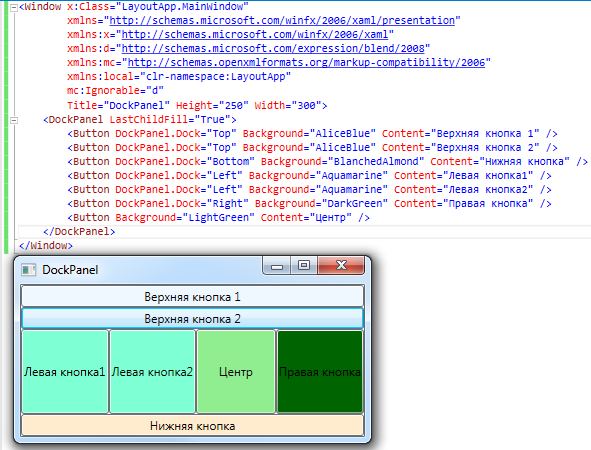


Причем у последней кнопки мы можем не устанавливать свойство DockPanel.Dock. Она уже заполняет все оставшееся пространство. Такой эффект получается благодаря установке у DockPanel свойства LastChildFill="True", которое означает, что последний элемент заполняет все оставшееся место. Если у этого свойства поменять True на False, то кнопка прижмется к левой стороне, заполнив только о место, которое ей необходимо.

Также обратите внимание на порядок прикрепления к кнопкам свойства DockPanel.Dock. Например, если мы изменим порядок нижняя кнопка уже будет заполнять меньшее место.



Также можно прижать к одной стороне сразу несколько элементов. В этом случае они просто будут располагаться по порядку:



Контейнер DockPanel особенно удобно использовать для создания стандартных интерфейсов, где верхнюю и левую часть могут занимать какие-либо меню, нижнюю - строка состояния, правую - какая-то дополнительная информация, а в центре будет находиться основное содержание.

**Элементы управления WPF**

Чтобы как-то взаимодействовать с пользователем, получать от пользователя ввод с клавиатуры или мыши и использовать введенные данные в программе, нам нужны элементы управления. WPF предлагает нам богатый стандартный набор элементов управления

Все элементы управления могут быть условно разделены на несколько подгрупп:

* Элементы управления содержимым, например кнопки (Button), метки (Label)
* Специальные контейнеры, которые содержат другие элементы, но в отличие от элементов Grid или Canvas не являются контейнерами компоновки - ScrollViewer,GroupBox
* Декораторы, чье предназначение создание определенного фона вокруг вложенных элементов, например, Border или Viewbox.
* Элементы управления списками, например, ListBox, ComboBox.
* Текстовые элементы управления, например, TextBox, RichTextBox.
* Элементы, основанные на диапазонах значений, например, ProgressBar, Slider.
* Элементы для работ с датами, например, DatePicker и Calendar.
* Остальные элементы управления, которые не вошли в предыдущие подгруппы, например, Image.

Все элементы управления наследуются от общего класса System.Window.Controls.Control и имеют ряд общих свойств.

**Иерархии WPF**

**System.Threading.DispatcherObject**

В основе WPF лежит модель STA (Single-Thread Affinity), согласно которой за пользовательский интерфейс отвечает один поток. И чтобы пользовательский интерфейс мог взаимодействовать с другими потоками, WPF использует концепцию диспетчера - специального объекта, управляющего обменом сообщениями, через которые взаимодействуют потоки. Наследование типов от класса DispatcherObject позволяет получить доступ к подобному объекту-диспетчеру и и другим функциям по управлению параллелизмом.

**System.Windows.DependencyObject**

Наследование от этого класса позволяет взаимодействовать с элементами в приложении через их специальную модель свойств, которые называются свойствами зависимостей (dependency properties). Эта модель упрощает применение ряда особенностей WPF, например, привязки данных. Так, система свойств зависимостей отслеживает зависимости между значениями свойств, автоматически проверяет их и изменяет при изменении зависимости.

**System.Windows.Media.Visual**

Класс Visual содержит инструкции, которые отвечают за отрисовку, визуализацию объекта.

**System.Windows.UIElement**

Класс UIElement добавляет возможности по компоновке элемента, обработку событий и получение ввода.

**System.Windows.FrameworkElement**

Класс FrameworkElement добавляет поддержку привязки данных, анимации, стилй. Также добавляет ряд свойств, связанных с компоновкой (выравнивание, отступы) и ряд других.

**System.Windows.Controls.Control**

Класс Control представляет элемент управления, с которым взаимодействует пользователь. Этот класс добавляет ряд дополнительных свойств для поддержки элементами шрифтов, цветов фона, шрифта, а также добавляет поддержку шаблонов - специального механизма в WPF, который позволяет изменять стандартное представление элемента, кастомизировать его.

И далее от класса Control наследуются непосредственно конкретные элементы управления или их базовые классы, которые получают весь функционал, добавляемый к типам в этой иерархии классов.

Рассмотрим некоторые из основных свойств, которые наследуются элементами управления.

**Позиционирование контента**

**Content Alignment**

Выравнивание содержимого внутри элемента задается свойствами **HorizontalContentAlignment** (выравнивание по горизонтали) и **VerticalContentAlignment** (выравнивание по вертикали), аналогичны свойствам VerticalAlignment/HorizontalAlignment. Свойство HorizontalContentAlignment принимает значения Left, Right, Center (положение по центру), Stretch (растяжение по всей ширине). Например:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <StackPanel>      <Button Margin="5" HorizontalContentAlignment="Left" Content="Left" Height="90" Width="500" />      <Button Margin="5" HorizontalContentAlignment="Right" Content="Right" Height="90" Width="500" />      <Button Margin="5" HorizontalContentAlignment="Center" Content="Center" Height="90" Width="500" />  </StackPanel> |

VerticalContentAlignment принимает значения Top (положение в верху), Bottom (положение внизу), Center (положение по центру), Stretch (растяжение по всей высоте)

**Padding**

С помощью свойства Padding мы можем установить отступ содержимого элемента:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <StackPanel>      <Button x:Name="button1" Padding="50 30 0 40" HorizontalContentAlignment="Left">          Hello World      </Button>      <Button x:Name="button2" Padding="60 20 0 30" HorizontalContentAlignment="Center">          Hello World      </Button>  </StackPanel> |

Свойство Padding задается в формате Padding="отступ\_слева отступ\_сверху отступ\_справа отступ\_снизу".

Если со всех четырех сторон предполагается один и тот же отступ, то, как и в случае с Margin, мы можем задать одно число:

|  |  |
| --- | --- |
| 1 | <Button x:Name="button2" Padding="20"  Content="Hello World" /> |

Важно понимать, от какой точки задается отступ. В случае с первой кнопкой в ней контект выравнивается по левому краю, поэтому отступ слева будет предполагать отступ от левого края элемента Button. А вторая кнопка располагается по центру. Поэтому для нее отступ слева предполагает отступ от той точки, в которой содержимое бы находилось при центрировании без применения Padding.

Комбинация значений свойств HorizontalContentAlignment/VerticalContentAlignment и Padding позволяет оптимальным образом задать расположение содержимого.

**Name**

Наверное важнейшее свойство. По установленному имени впоследствии можно будет обращаться к элементу, как в коде, так и в xaml разметке. Например, в xaml-коде у нас определена следующая кнопка:

<Button x:Name="button1" Width="60" Height="30" Content="Текст" Click="button1\_Click" />

Здесь у нас задан атрибут Click с названием метода обработчика button1\_Click, который будет определен в файле кода C# и будет вызываться по нажатию кнопки. Тогда в связанном файле кода C# мы можем обратиться к этой кнопке:

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

{

    button1.Content = "Привет!";

}

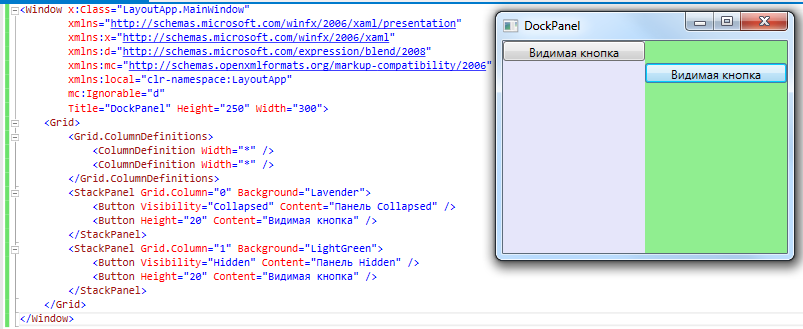
Поскольку свойство Name имеет значение button1, то через это значение мы можем обратиться к кнопке в коде.

**Visibility**

Это свойство устанавливает параметры видимости элемента и может принимать одно из трех значений:

* Visible - элемент виден и участвует в компоновке.
* Collapsed - элемент не виден и не участвует в компоновке.
* Hidden - элемент не виден, но при этом участвует в компоновке.

Hазличия между Collapsed и Hidden можно продемонстрировать на примере:



**Свойства настройки шрифтов**

* FontFamily - определяет семейство шрифта (например, Arial, Verdana и т.д.)
* FontSize - определяет высоту шрифта
* FontStyle - определяет наклон шрифта, принимает одно из трех значений - Normal, Italic,Oblique.
* FontWeight - определяет толщину шрифта и принимает ряд значений, как Black,Bold и др.
* FontStretch - определяет, как будет растягивать или сжимать текст, например, значение Condensed сжимает текст, а Expanded - расстягивает.

Например:

<Button Content="Hello World!" FontFamily="Verdana" FontSize="13" FontStretch="Expanded" />

**Cursor**

Это свойство позволяет нам получить или установить курсор для элемента управления в одно из значений, например, Hand, Arrow, Wait и др. Например, установка курсора в коде c#: button1.Cursor=Cursors.Hand;

**Цвет фона и шрифта**

Свойства Background и Foreground задают соответственно цвет фона и текста элемента управления.

Простейший способ задания цвета в коде xaml: Background="#ffffff". В качестве значения свойство Background (Foreground) может принимать запись в виде шестнадцатеричного значения в формате #rrggbb, где rr - красная составляющая, gg - зеленая составляющая, а bb - синяя. Также можно задать цвет в формате #aarrggbb.

Либо можно использовать названия цветов напрямую:

<Button Width="60" Height="30" Background="LightGray" Foreground="DarkRed" Content="Цвет" />

Однако при компиляции будет создаваться объект SolidColorBrush, который и будет задавать цвет элемента. То есть определение кнопки выше фактически будет равноценно следующему:

<Button Width="60" Height="30" Content="Цвет">

    <Button.Background>

        <SolidColorBrush Color="LightGray" />

    </Button.Background>

    <Button.Foreground>

        <SolidColorBrush Color="DarkRed" />

    </Button.Foreground>

</Button>

SolidColorBrush представляет собой кисть, покрывающую элемент одним цветом. Позже мы подробнее поговорим о цветах. А пока надо знать, что эти записи эквивалентны, кроме того, вторая форма определения цвета позволяет задать другие кисти - например, градиент.

Это надо также учитывать при установке или получении цвета элемента в коде c#:

button1.Background = new SolidColorBrush(Colors.Red);

button1.Foreground = new SolidColorBrush(Color.FromRgb(0,255, 0));

Класс Colors предлагает ряд встроенный цветовых констант, которыми мы можем воспользоваться. А если мы захотим конкретизировать настройки цвета с помощью значений RGB, то можно использовать метод Color.FromRgb.

[Элементы управления в WPF (webdelphi.ru)](https://csharp.webdelphi.ru/elementy-upravleniya-v-wpf/?ysclid=lam4uux747908752779" \l "i-5)