

УП МДК 02

Практическая работа 2

Элементы управления

Обзор элементов управления и их свойств

Теория:

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 - ScrollView, 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.Windows.Controls.Control и имеют ряд общих свойств.

Button

Элемент Button представляет обычную кнопку:

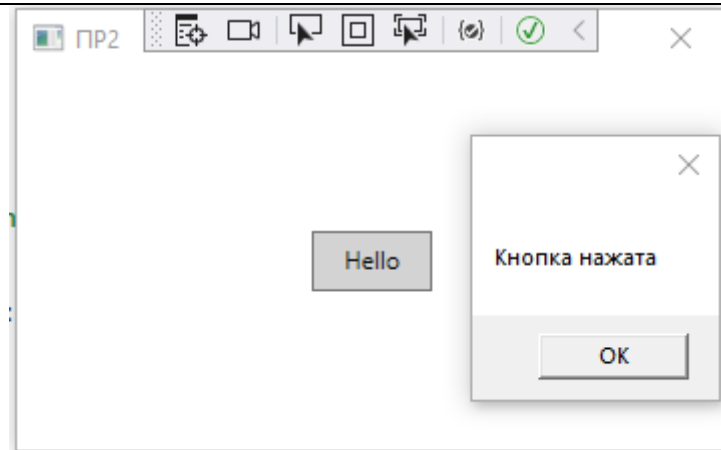
XAML

```
<Grid>
  <Button x:Name="button1" Width="60" Height="30" Background="LightGray"
    Content="Hello" Click="button1_Click"/>
</Grid>
```

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Кнопка нажата");
}
```

Результат



Отличительные особенности:

- Событие Click – нажатие на кнопку. В атрибуте Click указывается название функции-обработчика этого события.

- Свойство IsCancel. Возможные значения: True, False. Если записано True, то кнопка будет срабатывать при нажатии на кнопку Esc в данном окне, т.е. когда пользователь хочет закрыть окно без выполнения каких-либо действий.

- Свойство IsDefault. Возможные значения: True, False. Если записано True, то кнопка будет срабатывать при нажатии на кнопку Enter в данном окне, но только если не выделена какая-либо другая кнопка.

В отличие от приложения Windows Forms, в WPF-приложении при открытии окна не происходит автоматического выделения какого-либо элемента. Чтобы выделить первый элемент в окне, необходимо нажать кнопку Tab. Кнопка со свойством IsDefault="True" подсвечивается в окне, как будто она получила фокус. Но на самом деле кнопка не получает фокус, т.к. нажатие на клавишу «Пробел» не приводит к нажатию кнопки, а нажатие клавиши Tab приводит к выделению первого элемента на странице, а не элемента, следующего за кнопкой.

Элемент управления ToggleButton (переключаемая кнопка)

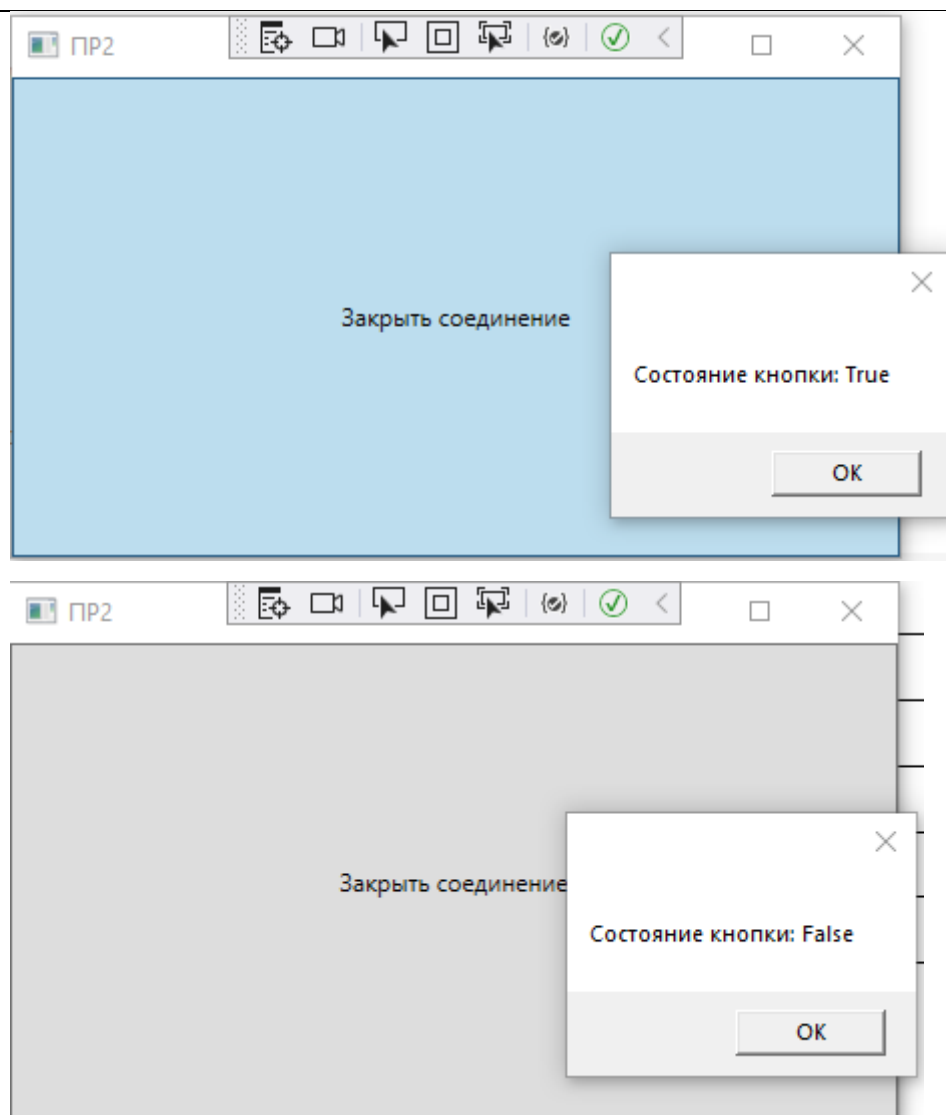
Представляет собой кнопку, которая может находиться в двух состояниях: нажатом и отжатом.

XAML

```
<Grid>
  <ToggleButton Click ="ToggleButton_Click">Закрыть соединение</ToggleButton>
</Grid>
```

C#

```
private void ToggleButton_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Состояние кнопки: " + (sender as System.Windows.Controls.Primitives.ToggleButton).IsChecked)
}
```

Результат

Отличительные особенности:

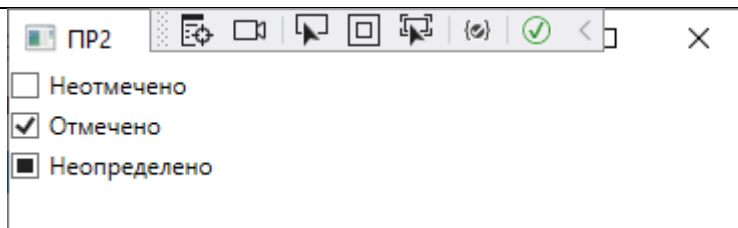
- Событие Click – нажатие или отжатие кнопки. В атрибуте Click указывается название функции обработчика этого события.
- Событие Checked – нажатие кнопки. В атрибуте Checked указывается название функции-обработчика этого события.
- Событие Unchecked – отжатие кнопки. В атрибуте Unchecked указывается название функции обработчика этого события.
- Свойство IsChecked – состояние кнопки. True – кнопка нажата, False – кнопка отжата

Элемент управления CheckBox (независимый переключатель)

XAML

```
<Grid>
  <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"/>
  </StackPanel>
</Grid>
```

Результат



Класс `CheckBox` является наследником от класса `ToggleButton` и наследует его свойства и события. Для обращения к элементу управления из кода программы необходимо в XAML-коде задать для него имя в атрибуте `Name` с префиксом `x`, как это показано в примере выше. Префикс `'x:'` означает пространство имен XAML, а не пространство имен WPF.

Элемент управления RadioButton (зависимый переключатель)

Элемент управления, также производный от `ToggleButton`, представляющий переключатель. Главная его особенность - поддержка групп.

Несколько элементов `RadioButton` можно объединить в группы, и в один момент времени мы можем выбрать из этой группы только один переключатель. Например,

XAML

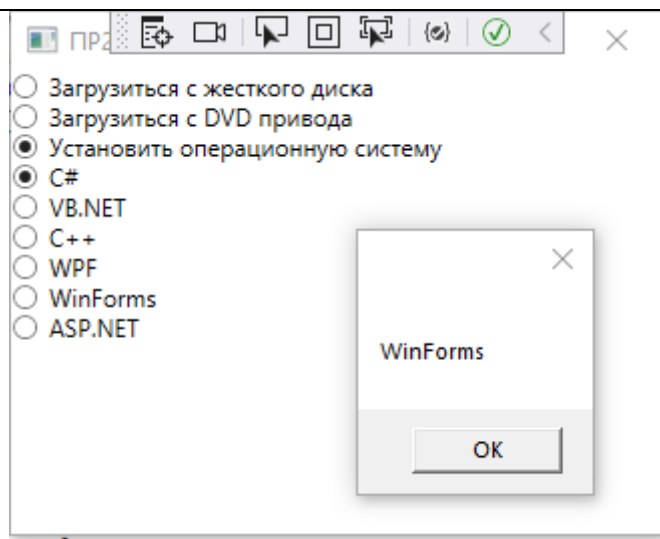
```
<Grid>
  <StackPanel x:Name="stackPanel">
    <RadioButton GroupName="Boot" x:Name="RadioButton_Boot1" Checked="RadioButton_Checked">Загрузиться с жесткого диска</RadioButton>
    <RadioButton GroupName="Boot" x:Name="RadioButton_Boot2" Checked="RadioButton_Checked">Загрузиться с DVD привода</RadioButton>
    <RadioButton GroupName="Boot" x:Name="RadioButton_Boot3" Checked="RadioButton_Checked">Установить операционную систему</RadioButton>
    <RadioButton GroupName="Languages" Content="C#" IsChecked="True" Checked="RadioButton_Checked"/>
    <RadioButton GroupName="Languages" Content="VB.NET" Checked="RadioButton_Checked"/>
    <RadioButton GroupName="Languages" Content="C++" Checked="RadioButton_Checked"/>
    <RadioButton GroupName="Technologies" Content="WPF" IsChecked="True" Checked="RadioButton_Checked"/>
    <RadioButton GroupName="Technologies" Content="WinForms" Checked="RadioButton_Checked"/>
    <RadioButton GroupName="Technologies" Content="ASP.NET" Checked="RadioButton_Checked"/>
  </StackPanel>
</Grid>
```

C#

Ссылка: 9

```
private void RadioButton_Checked(object sender, RoutedEventArgs e)
{
    RadioButton pressed = (RadioButton)sender;
    MessageBox.Show(pressed.Content.ToString());
}
```

Результат



Вопрос: почему при старте программы выходят сообщения C# и WPF?

Элемент управления `ComboBox` (выпадающий список)

Элемент `ComboBox` представляет собою выпадающий список, элементы которого определены с помощью элементов `ComboBoxItem`:

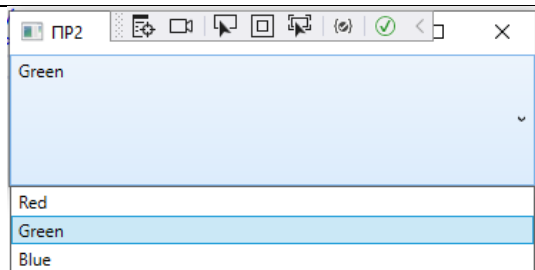
XAML

```

<Grid>
  <ComboBox SelectedIndex="1">
    <ComboBoxItem Content="Red" />
    <ComboBoxItem Content="Green" />
    <ComboBoxItem Content="Blue" />
  </ComboBox>
</Grid>

```

Результат



В качестве содержимого элементов выпадающего списка можно задавать не только текст, но и другие элементы, например эллипс или прямоугольник.

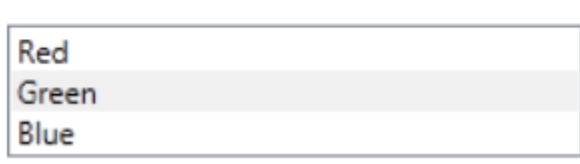
Элемент управления ListBox (список)

Элемент ListBox представляет собою список, элементы которого определены с помощью элементов ListBoxItem:

```

<ListBox SelectedIndex="1">
  <ListBoxItem Content="Red" />
  <ListBoxItem Content="Green" />
  <ListBoxItem Content="Blue" />
</ListBox>

```



В качестве содержимого элементов списка можно задавать не только текст, но и другие элементы.

После заполнения элемента управления ComboBox (или ListBox) есть три способа определить выбранного в них элемента. Во-первых, если необходимо найти числовой индекс выбранного элемента, необходимо использовать свойство SelectedIndex (отсчет начинается с 0; -1 означает

отсутствие выбора). Во-вторых, если требуется получить объект, выбранный внутри списка, то используется свойство `SelectedItem`. В-третьих, `SelectedValue` позволяет получить значение выбранного объекта

Элемент управления Slider

Элемент `Slider` представляет собою ползунок с минимальным значением `Minimum`, максимальным значением `Maximum` и текущим значением `Value`.

```
<Slider Height="25" Width="100" Minimum="1" Maximum="100" Value="20" />
```

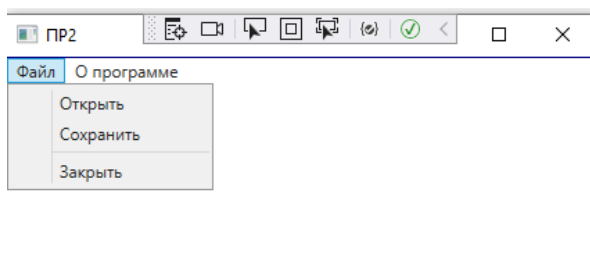


Меню

Меню в WPF представлено классом `Menu`, который может включать в себя набор объектов `MenuItem`. Каждый объект `MenuItem` в свою очередь может включать в себя другие объекты `MenuItem` и объекты `Separator` (разделитель). Пример элемента `Menu`:

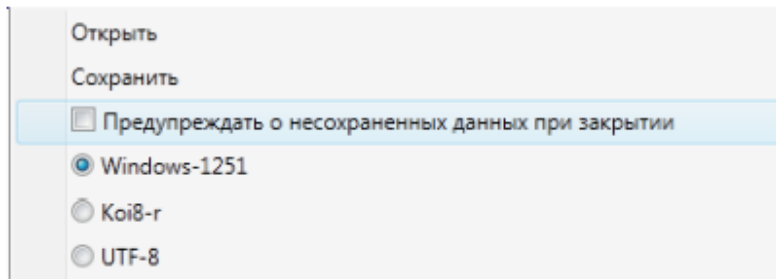
```
<Menu Background="White" BorderBrush="Navy" BorderThickness="1">
  <MenuItem Header="_Файл">
    <MenuItem Header="_Открыть" />
    <MenuItem Header="_Сохранить" />
    <Separator />
    <MenuItem Header="_Закрыть" />
  </MenuItem>
  <MenuItem Header="_О программе" />
</Menu>
```

Знак подчеркивания в названиях пунктов меню указывает «горячие» клавиши для доступа к этим пунктам меню. Пример работы приложения:



Элемент MenuItem может содержать и другие элементы управления, например зависимые (RadioButton) и независимые (CheckBox) переключатели:

```
<CheckBox Content="Предупредить о несохраненных данных при закрытии" />
<RadioButton GroupName="codepage" Content="Windows-1251" />
<RadioButton GroupName="codepage" Content="Koi8-r" />
<RadioButton GroupName="codepage" Content="UTF-8" />
```

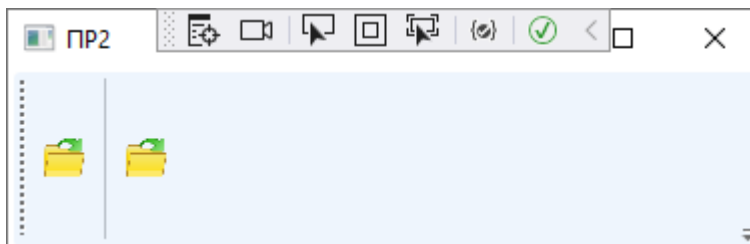


Панель инструментов

Панель инструментов в WPF представлена классом ToolBar, который в качестве содержимого может включать в себя коллекцию любых других элементов. Панели инструментов обычно используются как альтернативный способ активизации пунктов меню. Пример элемента ToolBar:

```
<ToolBar>
  <Button>
    <Image Width="30" Height="25" Source="https://img.cr4chrome.com/55/be/e1/bc-logo.png"></Image>
  </Button>
  <Separator/>
  <Button>
    <Image Width="30" Height="25" Source="D:\open.png" />
  </Button>
</ToolBar>
```

Пример работы приложения:



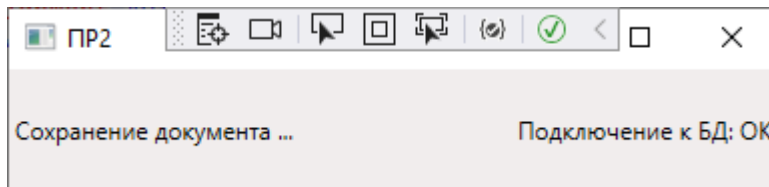
Строка состояния

Строка состояния в WPF представлена классом StatusBar, который в качестве содержимого может включать в себя коллекцию любых других элементов, в том числе StatusBarItem

Пример элемента StatusBar:

```
<StatusBar DockPanel.Dock="Bottom">
    <TextBlock Text="Сохранение документа ..." />
    <StatusBarItem HorizontalAlignment="Right" >
        <TextBlock Text="Подключение к БД: ОК" />
    </StatusBarItem>
</StatusBar>
```

Пример работы приложения:



Элемент TextBlock может применяться для отображения текста с добавлением форматирования: полужирный текст, подчеркнутый текст, разрывы строк и т.д

Элемент управления InkCanvas

Элемент управления InkCanvas позволяет рисовать и редактировать линии с помощью мыши или пера. Размеры элемента управления можно задать с помощью свойств Width и Height. Свойства пера (цвет, ширину и высоту) можно настроить с помощью свойства DefaultDrawingAttributes:

```
<InkCanvas>
    <InkCanvas.DefaultDrawingAttributes>
        <DrawingAttributes Color="Red" Height="10" Width="1"/>
    </InkCanvas.DefaultDrawingAttributes>
</InkCanvas>
```

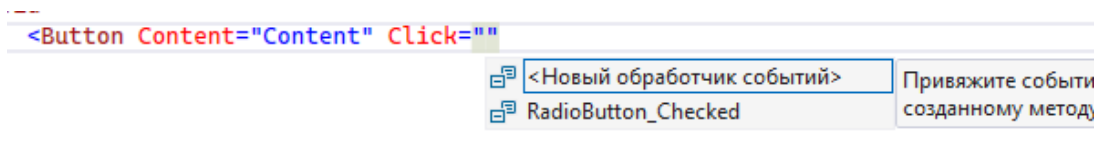
Результат выполнения данного участка программы:



Свойство EditingMode позволяет настроить режим редактирования: рисование (Ink), выбор и редактирование фигур (Select), удаление по точкам (EraseByPoint) и удаление фигур (EraseByStroke).

Обработчики событий

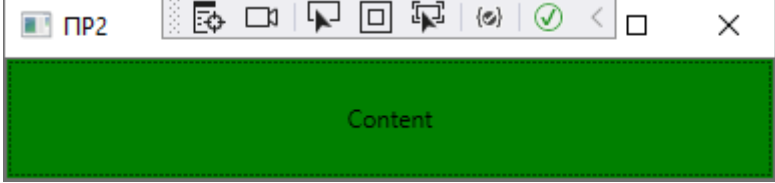
Для добавления обработчика для какого-либо события объекта необходимо в открывающем теге элемента написать имя события и через знак «=» имя функции-обработчика, либо выбрать команду «Новый обработчик события»:



При выборе команды «Новый обработчик события» в CS-файле, относящемся к XAML-файлу, будет добавлена соответствующая функция:

```
Ссылка: 0
private void Button_Click(object sender, RoutedEventArgs e)
{
    ...
}
```

В обработчике можно обратиться по имени к любому объекту, для которого в XAML-файле было определено имя с помощью атрибута Name или x:Name:

XAML
<code><Button x:Name="Button1" Content="Content" Click="Button_Click"/></code>
C#
<pre>private void Button_Click(object sender, RoutedEventArgs e) { Button1.Background = Brushes.Green; }</pre>
Результат


С помощью объекта sender, переданного в качестве параметра, можно получить доступ к элементу управления, для которого возникло обрабатываемое событие, даже в случае, если для него не задано имя:

```
private void CheckBox_Click(object sender, RoutedEventArgs e)
{
    ((FrameworkElement)sender).Visibility = System.Windows.Visibility.Hidden;
}

private void CheckBox_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show(((CheckBox)sender).IsChecked.ToString());
}
```

В первом примере объект sender был приведен к базовому классу FrameworkElement для доступа к базовым свойствам, присущим всем элементам управления. Во втором случае объект sender был приведен к классу CheckBox для доступа к специфическим свойствам данного элемента управления.

Если для нескольких элементов управления определен один обработчик какого-либо события, то для определения выбранного элемента управления в коде обработчика можно использовать свойство Tag, доступное для всех элементов управления:

```
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
    if (((FrameworkElement)sender).Tag.ToString() == "open") MessageBox.Show("Выбрана команда 'Открыть'");
    else
    if (((FrameworkElement)sender).Tag.ToString() == "save") MessageBox.Show("Выбрана команда 'Сохранить'");
}
```

Наиболее часто используемые события:

Click Происходит при нажатии на элемент управления

MouseMove Происходит, когда указатель мыши совершает движение по этому элементу

MouseEnter Происходит, когда указатель мыши входит в границы данного элемента

MouseLeave Происходит, когда указатель мыши покидает границы данного элемента

MouseDown Происходит при нажатии кнопки мыши, если указатель мыши находится на элементе

MouseUp Происходит, когда кнопка мыши отпускается на элементе

MouseWheel Происходит при прокрутке пользователем колесика мыши, если указатель мыши находится на элементе

KeyDown Происходит при нажатии клавиши, если элемент имеет фокус

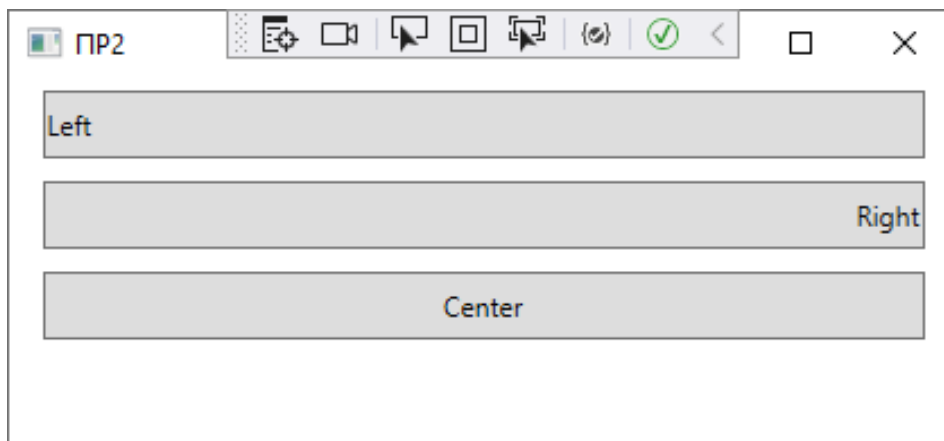
KeyUp Происходит при отжатии клавиши, если элемент имеет фокус

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:

```
Title="ПР2" Height="200" Width="400">
<Grid>
  <StackPanel>
    <Button Margin="5" HorizontalContentAlignment="Left" Content="Left" Height="30" Width="390" />
    <Button Margin="5" HorizontalContentAlignment="Right" Content="Right" Height="30" Width="390" />
    <Button Margin="5" HorizontalContentAlignment="Center" Content="Center" Height="30" Width="390" />
  </StackPanel>
</Grid>
```

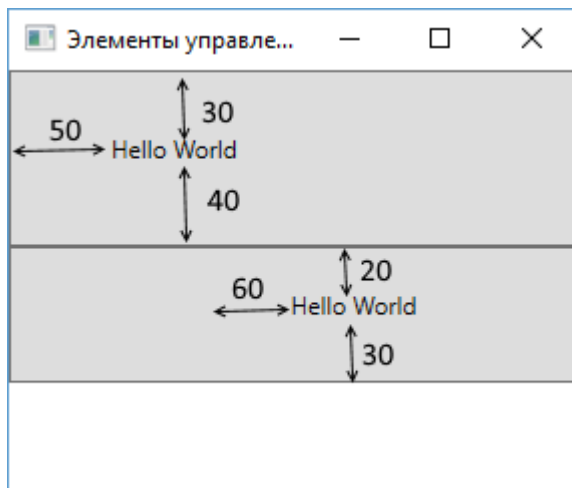


`VerticalContentAlignment` принимает значения `Top` (положение вверху), `Bottom` (положение внизу), `Center` (положение по центру), `Stretch` (растяжение по всей высоте)

Padding

С помощью свойства `Padding` мы можем установить отступ содержимого элемента:

```
<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:

```
<Button x:Name="button2" Padding="20" Content="Hello World" />
```

Важно понимать, от какой точки задается отступ. В случае с первой кнопкой в ней контекст 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 `HorizontalAlignment/VerticalContentAlignment` и `Padding` позволяет оптимальным образом задать расположение содержимого.

Цвета 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>
```

С помощью таких свойств, как FontFamily, TextDecorations и др., мы можем настроить отображение текста.

```
<Button Width="60" Height="30" Content="Цвет" FontSize="20" Foreground="Red" FontWeight="Bold"/>
```



Элементы Run представляют куски обычного текста, для которых можно задать отдельное форматирование.

Для изменения параметров отображаемого текста данный элемент имеет такие свойства, как LineHeight, TextWrapping и TextAlignment.

Свойство LineHeight позволяет указывать высоту строк.

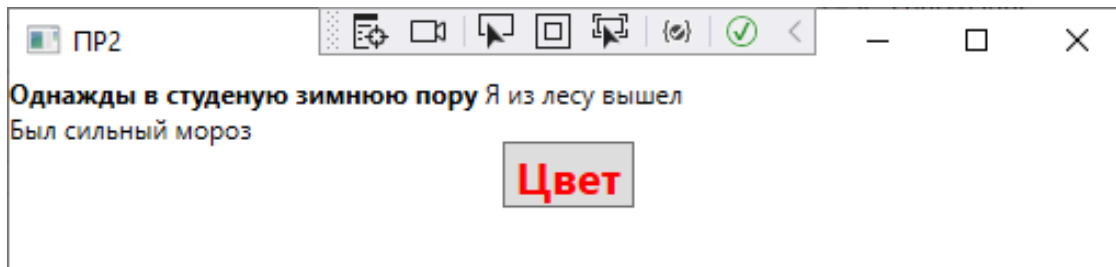
Свойство TextWrapping позволяет переносить текст при установке этого свойства TextWrapping="Wrap". По умолчанию это свойство имеет значение NoWrap, поэтому текст не переносится.

Свойство TextAlignment выравнивает текст по центру (значение Center), правому (Right) или левому краю (Left): <TextBlock TextAlignment="Right">

Для декорации текста используется свойство TextDecorations, например, если TextDecorations="Underline", то текст будет подчеркнут.

Если нам вдруг потребуется перенести текст на другую строку, то тогда мы можем использовать элемент LineBreak:

```
<TextBlock>
<Run FontWeight="Bold">Однажды в студеную зимнюю пору</Run> Я из лесу вышел <LineBreak /> Был сильный мороз </TextBlock>
```



Задание 1.

1. Создайте проект с названием Wpf_FIO_PR2, где FIO – ваша фамилия.
2. Разработать WPF-приложение с меню, панелью инструментов и строкой состояния.
3. С помощью пунктов меню пользователь может изменять цвет фона окна, получить информацию о разработчике, а также закрыть окно.
4. Кнопки панели инструментов дублируют команды меню.
5. При наведении на пункты меню или кнопки панели инструментов в строке состояния отображается информация об этих элементах управления.

Задание 2

Добавьте к прошлому заданию кнопку перехода на задание 2(новое окно).

Разработать «Графический редактор» с выпадающим списком для выбора цвета кисти, ползунком для выбора размеров кисти.

В качестве ластика использовать цвет фона.

В отчете предоставить:

- Титульный лист
- Экранные формы промежуточных этапов создания приложения
- Итоговые экранные формы задания 1 и 2
- Ссылку на GitHub