Лекция 5
Элементы управления WPF

Разработка программных модулей

Тимашева Эльза Ринадовна

1. Элементы управления. Обзор
2. Кнопки
3. CheckBox
4. RadioButton
5. ScrollViewer
6. Текстовые элементы управления:
6.1. TextBlock
6.2. TextBox
6.3. Label
6.4. PasswordBox
6.5. RichTextBox

Элементы управления

Чтобы как-то взаимодействовать с пользователем, получать от пользователя ввод с клавиатуры или мыши и использовать введенные данные в программе, нужны элементы управления.

WPF предлагает широкий 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.

Элементы управления

Рассмотрим некоторые из основных свойств, которые наследуются элементами управления.

Name

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

```
<Button x:Name="button" Width="60" Height="30" Content="Текст" Click="button_Click"></Button>
```

Здесь задан атрибут Click с названием метода обработчика button_Click, который будет определен в файле кода C# и будет вызываться по нажатию кнопки. Тогда в связанном файле кода C# можно обратиться к этой кнопке:

```
private void button_Click(object sender, RoutedEventArgs e)
    {
    }
```

Поскольку свойство Name имеет значение button, то через это значение можно 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.

Элементы управления

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** - растягивает.

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.
- Либо можно использовать названия цветов напрямую:

```
<Button Width="60" Height="30" Background="LightCyan" Foreground="Blue" Content="Цветная кнопка"></Button>
```

Элементы управления содержимым

К элементам управления содержимым относятся такие элементы как Button, Label, ToggleButton, ToolTip, RadioButton, CheckBox, GroupBox, TabItem, Expander, ScrollViewer. Также элементом управления содержимым является и главный элемент окна - Window.

Отличительной чертой всех этих элементов является наличие свойства **Content**, которое и устанавливает вложенный элемент. В этом элементы управления содержимым схожи с контейнерами компоновки. Только контейнеры могут иметь множество вложенных элементов, а элементы управления содержимым только один.

Рассмотрим на примере кнопки, которая является элементом управления содержимым:

```
<Button Content="Hello World!" ></Button>
```

В отличие от контейнеров компоновки для элементов управления содержимым можно задать только один вложенный элемент. Если же надо вложить в элемент управления содержимым несколько элементов, то можно использовать те же контейнеры компоновки:

```
<StackPanel>
    <TextBlock Text="Набор кнопкок" />
    <Button Background="Red" Height="20" Content="Red"></Button>
    <Button Background="Yellow" Height="20" Content="Yellow" ></Button>
    <Button Background="Green" Height="20" Content="Green" ></Button>
</StackPanel>
```

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:

```
<StackPanel>
 <Button Margin="5" HorizontalContentAlignment="Left" Content="Left" Height="90" Width="500"></Button>
 <Button Margin="5" HorizontalContentAlignment="Right" Content="Right" Height="90" Width="500"> </Button>
  <Button Margin="5" HorizontalContentAlignment="Center" Content="Center" Height="90" Width="500"> </Button>
</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 можно 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, можно задать одно число.

Кнопки
Button

Элемент Button представляет обычную кнопку:

```
<Button x:Name="button" Width="60" Height="30" Background="LightGray"></Button>
```

Чтобы связать кнопку с обработчиком события нажатия, нам надо определить в самой кнопке атрибут Click. А значением этого атрибута будет название обработчика в коде C#. А затем в самом коде C# определить этот обработчик.

К унаследованным свойствам кнопка имеет такие свойства как **IsDefault** и **IsCancel**, которые принимают значения true и false.

Если свойство IsDefault установлено в true, то при нажатии клавиши Enter будет вызываться обработчик нажатия этой кнопки.

Аналогично если свойство IsCancel будет установлено в true, то при нажатии на клавишу Esc будет вызываться обработчик нажатия этой кнопки.

Например, определим код xaml:

```
<StackPanel Orientation="Vertical">
  <Button x:Name="acceptButton" Content="OK" IsDefault="True" Click="acceptButton_Click"></Button>
  <Button x:Name="escButton" Content="Выход" IsCancel="True" Click="escButton_Click"></Button>
</StackPanel>
```

А в коде MainWindow.xaml.cs определим следующий код C#:

```
private void acceptButton_Click(object sender, RoutedEventArgs e)
   {
        MessageBox.Show("Действие выполнено");
   }
private void escButton_Click(object sender, RoutedEventArgs e)
   {
        this.Close(); // закрытие окна
   }
```

Теперь при нажатии на клавишу Enter будет отображаться сообщение, а при нажатии на Esc будет происходить выход из приложения и закрытие окна.

Кнопки

RepeatButton

Отличительная особенность элемента RepeatButton - непрерывная генерация события Click, пока нажата кнопка. Интервал генерации события корректируется свойствами **Delay** и **Interval**.

Сам по себе элемент RepeatButton редко используется, однако он может служить основой для создания ползунка в элементах ScrollBar и ScrollViewer, в которых нажатие на ползунок инициирует постоянную прокрутку.

ToggleButton

Представляет элементарный переключатель. Может находиться в трех состояниях - true, false и "нулевом" (неотмеченном) состоянии, а его значение представляет значение типа bool? в языке C#. Состояние можно установить или получить с помощью свойства **IsChecked**. Также добавляет три события - **Checked** (переход в отмеченное состояние), **Unchecked** (снятие отметки) и **Intermediate** (если значение равно null). Чтобы отрабатывать все три события, надо установить свойство **IsThreeState="True"**

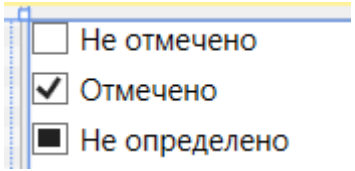
ToggleButton, как правило, сам по себе тоже редко используется, однако при этом он служит основой для создания других более функциональных элементов, таких как checkbox и radiobutton.

CheckBox

Элемент CheckBox представляет собой обычный флажок. Данный элемент является производным от класса ToggleButton и поэтому может принимать также три состояния: **Checked**, **Unchecked** и **Intermediate**.

Чтобы получить или установить определенное состояние, надо использовать свойство **IsChecked:**

```
<StackPanel x:Name="stackPanel">
  <CheckBox x:Name="checkBox1" IsThreeState="True" IsChecked="False" Height="20"
   Content="Не отмечено"></CheckBox>
  <CheckBox x:Name="checkBox2" IsThreeState="True" IsChecked="True" Height="20"
Content="Отмечено"></CheckBox>
  <CheckBox x:Name="checkBox3" IsThreeState="True" IsChecked="{x:Null}" Height="20" Content="Не
определено"></CheckBox>
</StackPanel>
```

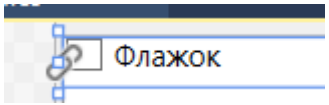


Кнопки

CheckBox

Ключевыми событиями флажка являются события **Checked** (генерируется при установке флажка в отмеченное состояние), **Unchecked** (генерируется при снятии отметки с флажка) и **Indeterminate** (флажок переведен в неопределенное состояние). Например, определим флажок:

```
<CheckBox x:Name="checkBox" IsChecked="False" Height="20" Content="Флажок"
    IsThreeState="True"
    Unchecked="checkBox_Unchecked"
    Indeterminate="checkBox_Indeterminate"
    Checked="checkBox_Checked"></CheckBox>
```



А в файле кода C# пропишем для него обработчики:

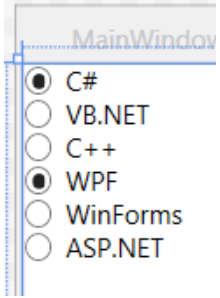
```
private void checkBox_Unchecked(object sender, RoutedEventArgs e)
  {
      MessageBox.Show(checkBox.Content.ToString() + " не отмечен");
  }
private void checkBox_Indeterminate(object sender, RoutedEventArgs e)
  {
      MessageBox.Show(checkBox.Content.ToString() + " в неопределенном состоянии");
  }
private void checkBox_Checked(object sender, RoutedEventArgs e)
  {
      MessageBox.Show(checkBox.Content.ToString() + " отмечен");
  }
```

Кнопки

RadioButton

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 можно выбрать из этой группы только один переключатель. Например,

```
<StackPanel x:Name="stackPanel">
  <RadioButton GroupName="Languages" Content="C#" IsChecked="True"></RadioButton>
  <RadioButton GroupName="Languages" Content="VB.NET"></RadioButton>
  <RadioButton GroupName="Languages" Content="C++"></RadioButton>
  <RadioButton GroupName="Technologies" Content="WPF" IsChecked="True"></RadioButton>
  <RadioButton GroupName="Technologies" Content="WinForms"></RadioButton>
  <RadioButton GroupName="Technologies" Content="ASP.NET"></RadioButton>
</StackPanel>
```



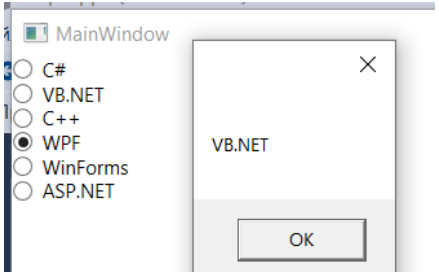
Чтобы включить элемент в определенную группу, используется свойство GroupName. В данном случае у нас две группы - Languages и Technologies. Можно отметить не более одного элемента RadioButton в пределах одной группы, зафиксировав тем самым выбор из нескольких возможностей.

Чтобы проследить за выбором того или иного элемента, также можно определить у элементов событие Checked и его обрабатывать в коде:

```
<RadioButton GroupName="Languages" Content="VB.NET" Checked="RadioButton_Checked"></RadioButton>
```

Обработчик в файле кода:

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



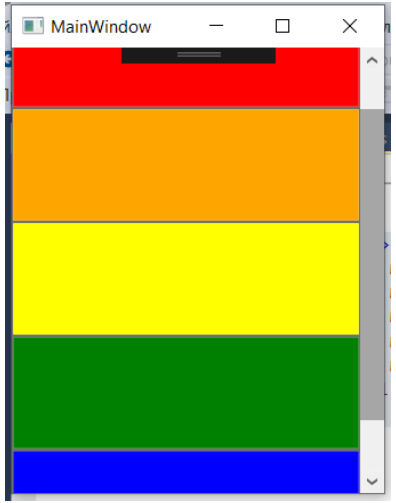
Всплывающие подсказки. Tooltip и Popup

Контейнеры GroupBox и Expander

ScrollViewer. Создание прокрутки

Элемент ScrollViewer обеспечивает прокрутку содержимого. Может вмещать в себя только один элемент, поэтому все элементы, помещаемые внутрь ScrollViewer необходимо облачить в еще один контейнер. Например:

```
<Grid>
   <ScrollViewer>
      <StackPanel>
          <Button MinHeight="80" Background="Red"></Button>
          <Button MinHeight="80" Background="Orange"></Button>
          <Button MinHeight="80" Background="Yellow"></Button>
          <Button MinHeight="80" Background="Green"></Button>
          <Button MinHeight="80" Background="Blue"></Button>
      </StackPanel>
   </ScrollViewer>
</Grid>
```



ScrollViewer поддерживает как вертикальную, так и горизонтальную прокрутку. Ее можно установить с помощью свойств **HorizontalScrollBarVisibility** и **VerticalScrollBarVisibility**. Эти свойства принимают одно из следующих значений:

- **Auto**: наличие полос прокрутки устанавливается автоматически
- **Visible**: полосы прокрутки отображаются в окне приложения
- **Hidden**: полосы прокрутки не видно, но прокрутка возможна с помощью клавиш клавиатуры
- **Disabled**: полосы прокрутки не используются, а сама прокрутка даже с помощью клавиатуры невозможна

Среди свойств нужно отметить еще **CanContentScroll**. Если оно установлено в True, то прокрутка осуществляется не на несколько пикселей, а к началу следующего элемента.

Текстовые элементы управления

TextBlock

Элемент предназначен для вывода текстовой информации, для создания простых надписей:

```
<StackPanel>
    <TextBlock>Текст1</TextBlock>
    <TextBlock Text="Текст2"></TextBlock>
</StackPanel>
```

Ключевым свойством здесь является свойство Text, которое задает текстовое содержимое. Причем в случае <TextBlock>Текст1</TextBlock> данное свойство задается неявно.

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

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 потребуется перенести текст на другую строку, то можно использовать элемент LineBreak:

```
<TextBlock>
    Однажды в студеную зимнюю пору
    <LineBreak />
    Я из лесу вышел
</TextBlock>
```

Текстовые элементы управления

TextBox

Если TextBlock просто выводит статический текст, то этот элемент представляет поле для ввода текстовой информации.

Он также, как и TextBlock, имеет свойства **TextWrapping**, **TextAlignment** и **TextDecorations**.

С помощью свойства **MaxLength** можно задать предельное количество вводимых символов.

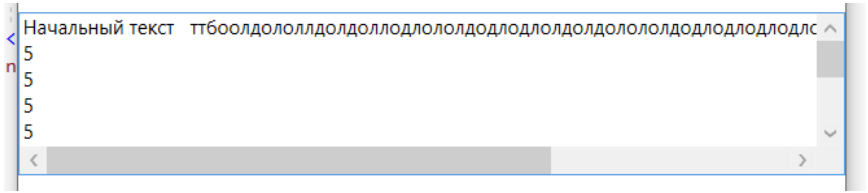
По умолчанию, если вводимый текст превышает установленные границы поля, то текстовое поле растет, чтобы вместить весь текст. Но визуально это не очень хорошо выглядит. Поэтому, как и в случае с TextBlock, мы можем перенести непомещающийся текст на новую строку, установив свойство TextWrapping="Wrap".

Чобы переводить по нажатию на клавишу Enter курсор на следующую строку, нам надо установить свойство AcceptsReturn="True".

Также мы можем добавить полю возможность создавать табуляцию с помощью клавиши Tab, установив свойство AcceptsTab="True"

Для отображения полос прокрутки TextBox поддерживает свойства **VerticalScrollBarVisibility** и **HorizontalScrollBarVisibility**:

```
<TextBox AcceptsReturn="True" Height="100"
   VerticalScrollBarVisibility="Auto"
   HorizontalScrollBarVisibility="Auto">
   Начальный текст</TextBox>
```

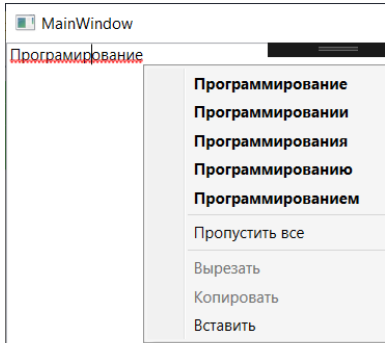


Возможно, при создании приложения нам потребуется сделать текстовое поле недоступным для ввода (на время в зависимости от условий или вообще), тогда для этого нам надо установить свойство IsReadOnly="True".

Для выделения текста есть свойства **SelectionStart**, **SelectionLength** и **SelectionText**.

TextBox обладает встроенной поддержкой орфографии. Чтобы ее задействовать, надо установить свойство SpellCheck.IsEnabled="True". Кроме того, по умолчанию проверка орфографии распространяется только на английский язык, поэтому, если приложение заточено под другой язык, нам надо его явным образом указать через свойство **Language:**

```
<TextBox SpellCheck.IsEnabled="True" Language="ru-ru">Програмирование</TextBox>
```



Текстовые элементы управления

Метка (Label)

Главной особенностью меток является поддержка мнемонических команд-клавиш быстрого доступа, которые передают фокус связанному элементу. Например,

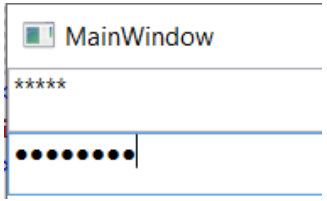
```
<Label Target="{Binding ElementName=TextBox1}">_привет</Label>
<TextBox Name="TextBox1" Margin="0 30 0 0" Height="30" Width="100"></TextBox>
```

Теперь, нажав на клавишу "п", мы переведем фокус на связанное текстовое поле. При вызове приложения подчеркивание не отображается, чтобы отображать подчеркивание, надо нажать на клавишу Alt. Тогда чтобы перевести фокус на связанное текстовое поле необходимо будет нажать сочетание Alt + "п". Если не предполагается использование клавиш быстрого доступа, то для вывода обычной текста вместо меток лучше использовать элемент TextBlock.

PasswordBox

Элемент предназначен для ввода парольной информации. По сути это тоже текстовое поле, только для ввода символов используется маска. Свойство **PasswordChar** устанавливает символ маски, отображаемый при вводе пароля. Если это свойство не задано, то по умолчанию для маски символа используется черная точка. Свойство **Password** устанавливает парольную строку, отображаемую по умолчанию при загрузке окна приложения.

```
<StackPanel>
   <PasswordBox PasswordChar="*" MinHeight="30"></PasswordBox>
   <PasswordBox MinHeight="30" ></PasswordBox>
</StackPanel>
```



RichTextBox

Для вывода текстового содержимого, насыщенного форматированием, графикой, предназначен RichTextBox. Можно даже сказать, что он выводит не просто текст, а документы с более сложным форматированием, чем обычный TextBox.

Текстовые элементы управления

Метка (Label)

Главной особенностью меток является поддержка мнемонических команд-клавиш быстрого доступа, которые передают фокус связанному элементу. Например,

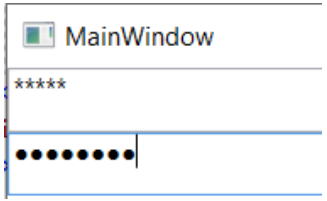
```
<Label Target="{Binding ElementName=TextBox1}">_привет</Label>
<TextBox Name="TextBox1" Margin="0 30 0 0" Height="30" Width="100"></TextBox>
```

Теперь, нажав на клавишу "п", мы переведем фокус на связанное текстовое поле. При вызове приложения подчеркивание не отображается, чтобы отображать подчеркивание, надо нажать на клавишу Alt. Тогда чтобы перевести фокус на связанное текстовое поле необходимо будет нажать сочетание Alt + "п". Если не предполагается использование клавиш быстрого доступа, то для вывода обычной текста вместо меток лучше использовать элемент TextBlock.

PasswordBox

Элемент предназначен для ввода парольной информации. По сути это тоже текстовое поле, только для ввода символов используется маска. Свойство **PasswordChar** устанавливает символ маски, отображаемый при вводе пароля. Если это свойство не задано, то по умолчанию для маски символа используется черная точка. Свойство **Password** устанавливает парольную строку, отображаемую по умолчанию при загрузке окна приложения.

```
<StackPanel>
   <PasswordBox PasswordChar="*" MinHeight="30"></PasswordBox>
   <PasswordBox MinHeight="30" ></PasswordBox>
</StackPanel>
```



RichTextBox

Для вывода текстового содержимого, насыщенного форматированием, графикой, предназначен RichTextBox. Можно даже сказать, что он выводит не просто текст, а документы с более сложным форматированием, чем обычный TextBox.