


Методические указания к лабораторной работе №3
«Основные элементы управления WPF»

Элемент управления *Button* (кнопка)

Представляет собой обычную кнопку.

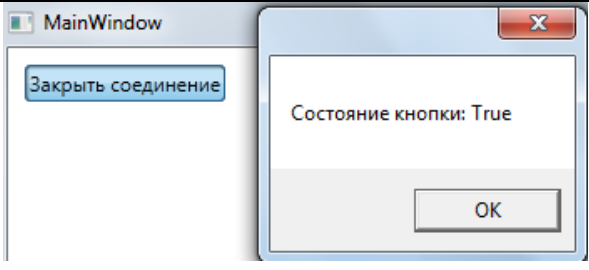
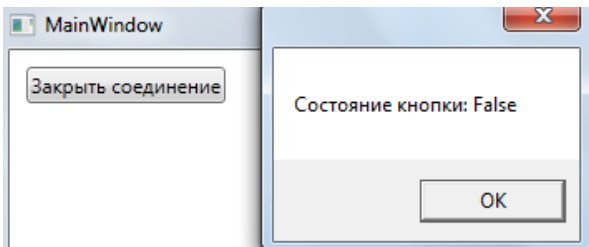
| Код XAML | Результат |
|--|--|
| <pre><Button Click="Button_Click">Отправить запрос</Button></pre> |  |
| Код C# | |
| <pre>private void Button_Click(object sender, RoutedEventArgs e) { MessageBox.Show("Кнопка нажата"); }</pre> | |

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.

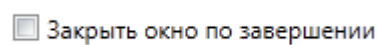
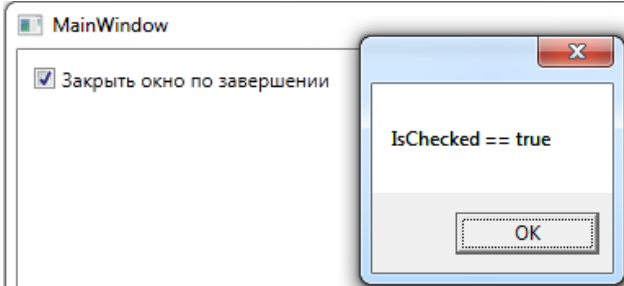
| Код XAML | Результат |
|---|--|
| <pre><ToggleButton Click="ToggleButton_Click">Закрыть соединение</ToggleButton></pre> |  |
| Код C# | |
| <pre>private void ToggleButton_Click(object sender, RoutedEventArgs e) { MessageBox.Show("Состояние кнопки: " + (sender as System.Windows.Controls.Primitives.ToggleButton).IsChecked); }</pre> |  |

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.

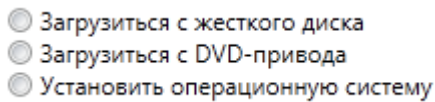
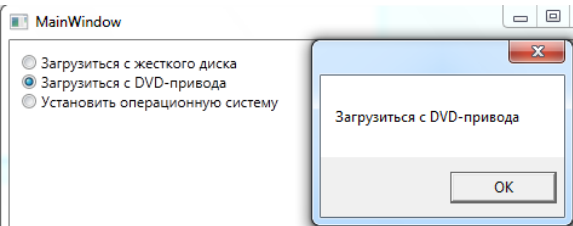
Элемент управления *CheckBox* (независимый переключатель)

| Код XAML | Результат |
|---|--|
| <pre><CheckBox x:Name="CheckBox_CloseAfterComplete">Закреть окно по завершении</CheckBox></pre> |  |
| Код C# | |
| <pre>... if (CheckBox_CloseAfterComplete.IsChecked == true) MessageBox.Show("IsChecked == true"); ...</pre> |  |

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)

| Код XAML | Результат |
|---|--|
| <pre><RadioButton GroupName="Boot" x:Name="RadioButton_Boot1">Загрузиться с жесткого диска</RadioButton> <RadioButton GroupName="Boot" x:Name="RadioButton_Boot2">Загрузиться с DVD- привода</RadioButton> <RadioButton GroupName="Boot" x:Name="RadioButton_Boot3">Установить операционную систему</RadioButton></pre> |  |
| Код C# | |
| <pre>... if (RadioButton_Boot1.IsChecked == true) MessageBox.Show(RadioButton_Boot1.Content.ToString()); else if (RadioButton_Boot2.IsChecked == true) MessageBox.Show(RadioButton_Boot2.Content.ToString()); else if (RadioButton_Boot3.IsChecked == true) MessageBox.Show(RadioButton_Boot3.Content.ToString()); ...</pre> |  |

Класс *RadioButton* является наследником от класса *ToggleButton* и наследует его свойства и события.

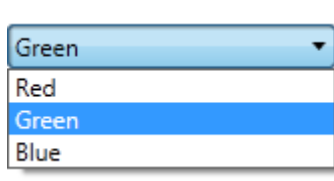
Отличительные особенности:

- Свойство **GroupName** – название группы зависимых переключателей. В одном окне может быть несколько групп зависимых переключателей с разными названиями групп.

Элемент управления *ComboBox* (выпадающий список)

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

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

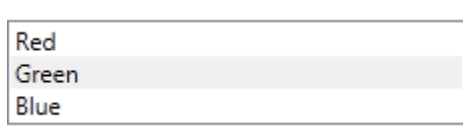


В качестве содержимого элементов выпадающего списка можно задавать не только текст, но и другие элементы, например эллипс или прямоугольник.

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">
```

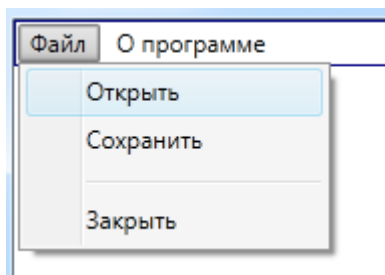
```

        <MenuItem Header="_Открыть" />
        <MenuItem Header="_Сохранить" />
        <Separator />
        <MenuItem Header="_Заккрыть" />
    </MenuItem>
    <MenuItem Header="_О программе" />
</Menu>

```

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

Пример работы приложения:

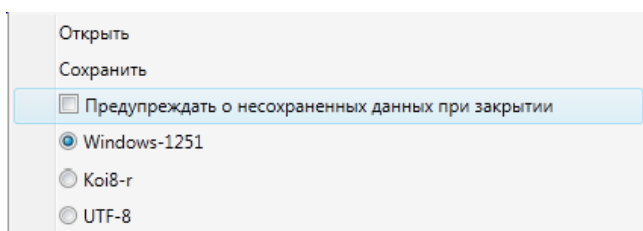


Элемент MenuItem может содержать и другие элементы управления, например зависимые (RadioButton) и независимые (CheckBox) переключатели:

```

<CheckBox Content="Предупреждать 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 Source="open.png"></Image>
    </Button>
    <Separator/>
    <Button>
        <Image Source="http://www.readyicons.com/IconSets/Sky_Light_%28Basic%29/48x48-save.png"></Image>
    </Button>
</ToolBar>

```

Пример работы приложения:



Кнопки содержат элементы Image. Первый элемент Image получает данные из файла open.png, включенного в проект. Второй элемент Image получает данные с веб-сайта по протоколу HTTP. Другие изображения можно выбрать, открыв в браузере адрес [http://www.readyicons.com/IconSets/Sky_Light %28Basic%29/](http://www.readyicons.com/IconSets/Sky_Light_%28Basic%29/)

Для создания нескольких панелей инструментов элементы ToolBar необходимо поместить в элемент ToolBarTray.

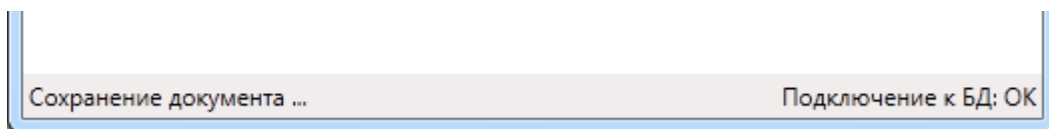
Строка состояния

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:



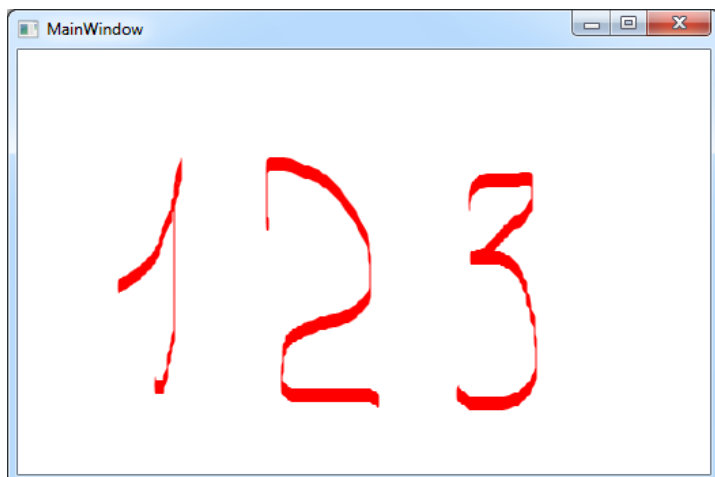
Элемент TextBlock может применяться для отображения текста с добавлением форматирования: полужирный текст, подчеркнутый текст, разрывы строк и т.д.

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:



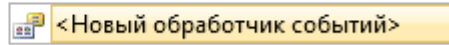
Свойство `EditMode` позволяет настроить режим редактирования: рисование (`Ink`), выбор и редактирование фигур (`Select`), удаление по точкам (`EraseByPoint`) и удаление фигур (`EraseByStroke`).

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»:

```
<MenuItem Header="_Открыть" Click="|" />
```

```
<MenuItem Header="_Сохранить" />
```



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

```
private void MenuItem_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`:

```
<MenuItem Name="mi_open" Header="_Открыть" Click="MenuItem_Click" />
```

```
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
    mi_open.Background = Brushes.LightGreen;
}
```

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

Задание 1

Разработать WPF-приложение с меню, панелью инструментов и строкой состояния. С помощью пунктов меню пользователь может изменять цвет фона окна, получить информацию о разработчике, а также закрыть окно. Кнопки панели инструментов дублируют команды меню. При наведении на пункты меню или кнопки панели инструментов в строке состояния отображается информация об этих элементах управления.

Задание 2

Разработать WPF-приложение «Графический редактор» с выпадающим списком для выбора цвета кисти, ползунком для выбора размеров кисти и зависимыми переключателями для выбора режима работы: «рисование», «редактирование», «удаление».