



**Разработка приложений  
с использованием WPF**

# Урок № 2

## Элементы управления

### Содержание

<b>1.</b>	<b>Модель управления содержимым .....</b>	<b>5</b>
1.1.	Вариации модели управления содержимым .....	7
<b>2.</b>	<b>Кисти заднего фона и переднего плана.....</b>	<b>10</b>
2.1.	Кисть SolidColorBrush.....	13
2.2.	Градиентные кисти.....	14
2.3.	Плиточные кисти .....	21
<b>3.</b>	<b>Шрифты.....</b>	<b>28</b>
3.1.	Размер шрифта.....	28
3.2.	Растяжение или сжатие текста.....	28
3.3.	Плотность текста.....	29
3.4.	Наклон текста.....	30
3.5.	Семейство шрифтов .....	30
3.6.	Наследование шрифтов .....	31
<b>4.</b>	<b>Примитивные элементы .....</b>	<b>32</b>
4.1.	Элемент TextBlock .....	33
4.2.	Элемент Image .....	36

4.3. Элемент MediaElement .....	38
4.4. Элемент Border.....	43
<b>5. Элемент управления Label .....</b>	<b>45</b>
<b>6. Группирующие элементы управления.....</b>	<b>48</b>
6.1. Элемент управления GroupBox.....	48
6.2. Элемент управления Expander .....	50
<b>7. Диапазонные элементы управления .....</b>	<b>54</b>
7.1. Элемент управления ProgressBar .....	56
7.2. Элемент управления Slider.....	57
7.3. Элемент управления ScrollBar.....	64
<b>8. Списковые элементы управления.....</b>	<b>67</b>
8.1. Элемент управления ListBox.....	69
8.2. Элемент управления ListView.....	72
8.3. Элемент управления ComboBox.....	72
8.4. Элемент управления TreeView .....	75
8.5. Элемент управления TabControl.....	80
<b>9. Всплывающие окна.....</b>	<b>83</b>
9.1. Позиционирование всплывающих окон .....	86
9.2. Взаимодействие свойств позиционирования между собой .....	100
9.3. Перекрытие всплывающего окна границами экрана.....	103
9.4. Настройка позиционирования всплывающего окна .....	106
<b>10.Меню .....</b>	<b>107</b>
10.1. Элемент управления Menu.....	107
10.2. Элемент управления ContextMenu.....	113
<b>11.Подсказки.....</b>	<b>118</b>

<b>12. Перетаскиваемые элементы управления .....</b>	<b>124</b>
12.1. Элемент управления GridSplitter .....	125
<b>13. Маршрутизация событий .....</b>	<b>132</b>
13.1. Прямая маршрутизация.....	133
13.2. Проверка на визуальное попадание .....	134
13.3. Параметры маршрутизируемых событий.....	137
<b>14. Обработка ввода .....</b>	<b>140</b>
14.1. Мышь .....	140
14.2. Клавиатура.....	145
<b>15. Обзор базовых классов элементов визуального дерева.....</b>	<b>154</b>
15.1. Класс UIElement.....	154
15.2. Класс FrameworkElement .....	156
15.3. Класс Control .....	158
<b>16. Домашнее задание .....</b>	<b>160</b>

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

# 1. Модель управления содержимым

Ранее, при рассмотрении базовых классов, использующихся для элементов, помещаемых в визуальное дерево, был упомянут класс `System.Windows.Controls.ContentControl`, который, в свою очередь, является наследником класса `System.Windows.Controls.Control`. Другими словами, это класс, описывающий более специализированный вид элементов управления. Его специализация заключается в том, что он способен содержать только один дочерний элемент, который помещается в свойство `System.Windows.Controls.ContentControl.Content`. Данное свойство обладает типом данных `System.Object`, что делает возможным присваивание ему объекта любого типа.

Когда элемент, обладающий содержимым, отображается на экране, то также происходит отображение и его содержимого. При этом, если объект установленный в качестве содержимого обладает типом, производным от `System.Windows.UIElement`, то он отображается согласно внешнему виду, присущему ему. Если же тип данных объекта-содержимого другой, то для него вызывается метод `System.Object.ToString`, который возвращает строковое представление объекта. Эта строка отображается в качестве содержимого.

В качестве примера давайте рассмотрим следующий фрагмент разметки:

## XAML

```
<Button Background="Green">
    <StackPanel Margin="5" Orientation="Horizontal">
        <Button>Nested Button</Button>
        <TextBox Margin="10,0,0,0" Width="100">Nested
            TextBox</TextBox>
    </StackPanel>
</Button>
```

Результат приведенной разметки показан на рис. 1.



Рис. 1. Свойство Content

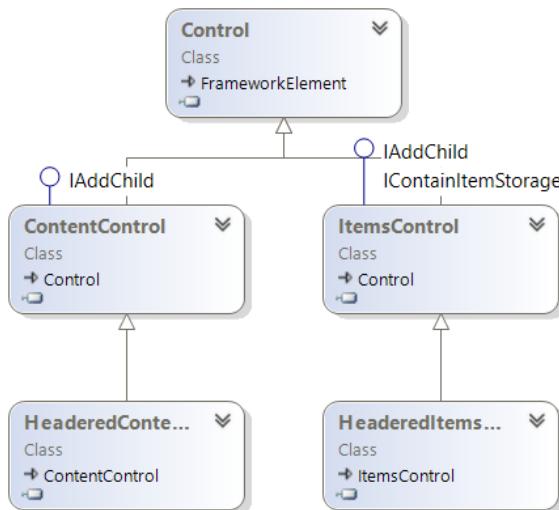
Такая модель управления содержимым предоставляет очень большие возможности при создании пользовательских интерфейсов. По началу может показаться, что элементы, которые могут содержать только один другой элемент в качестве содержимого слишком ограниченные для того, чтобы использовать их для построения сложных интерфейсов. Однако, это не так. Фактически можно обойти это ограничение, поместив в качестве дочернего элемента какую-нибудь панель. В этом случае можно будет добавить больше одного элемента в панель, так как все они поддерживают такую функциональность. Кстати, класс `System.Windows.Window`, описывающий окно верхнего уровня также является производным от класса `System.Windows.Controls.ContentControl` и также может содержать только один элемент в качестве содержимого. Именно поэтому, при создании нового окна в Visual Studio создается «заглушки», которая содержит окно и один дочерний элемент — панель, уже в которую помещаются все остальные элементы.

Свойства класса `System.Windows.Controls.ContentControl`:

- **Content** (`System.Object`). Получает или задает содержимое элемента. Значение по умолчанию — `null`.
- **HasContent** (`System.Boolean`). Получает значение, которое указывает, есть ли у элемента содержимое. `True` если у элемента есть содержимое; иначе — `false`.

## 1.1. Вариации модели управления содержимым

Рассмотренная выше модель управления содержимым является самой базовой. При этом, в WPF существует несколько основных вариаций данной модели (рис. 2).



**Рис. 2. Диаграмма классов, описывающих основные модели содержимого**

### 1.1.1. Класс `HeaderedContentControl`

Класс `System.Windows.Controls.HeaderedContentControl` расширяет класс `System.Windows.Controls.ContentControl`

заголовком, который используется производными классами для того, чтобы описывать содержимое элемента.

Для того, чтобы задать элементу заголовок необходимо использовать свойство `System.Windows.Controls.HeaderedContentControl.Header`, которое так же, как и свойство для указания содержимого обладает типом `System.Object`, что делает возможным помещение туда любого объекта. Принцип отображения для помещенных в заголовок объектов точно такой же, как и для свойства `System.Windows.Controls.ContentControl.Content`, т.е. либо преобразование объекта в строку и отображение этой строки, либо отображение, описанное элементом.

Свойства класса `System.Windows.Controls.HeaderedContentControl`:

- **HasHeader** (`System.Boolean`). Получает значение, которое указывает, есть ли у элемента заголовок. `True` если у элемента есть заголовок; иначе — `false`.
- **Header** (`System.Object`). Получает или задает заголовок элемента. Значение по умолчанию — `null`.

### 1.1.2. Класс `ItemsControl`

Класс `System.Windows.Controls.ItemsControl` является базовым для элементов управления, которые содержат список дочерних элементов, вместо одного. У этого класса свойством «содержимое» является свойство `System.Windows.Controls.ItemsControl.Items`, которое еще и является свойством-коллекцией. Это позволяет использовать упрощенный вариант разметки, не указывая явным образом название свойства и коллекцию, в которую помещаются дочерние элементы. Типом данных объектов коллекции

является тип `System.Object`, что позволяет помещать в нее любые объекты. При этом, если объект обладает типом, производным от `System.Windows.UIElement`, то он отображается согласно внешнему виду, присущему ему. Если же тип данных объекта, помещенного в коллекцию другой, то для него вызывается метод `System.Object.ToString`, который возвращает строковое представление объекта. Эта строка отображается в качестве элемента списка.

Свойства класса `System.Windows.Controls.ItemsControl`:

- **HasItems** (`System.Boolean`). Получает значение, которое указывает, содержит ли элемент хотя бы один объект `true` если элемент содержит хотя бы один объект; иначе — `false`.
- **Items** (`System.Windows.Controls.ItemCollection`). Получает коллекцию объектов, которые используются для генерирования содержимого элемента. Значение по умолчанию — пустая коллекция.

### **1.1.3. Класс HeaderedItemsControl**

Класс `System.Windows.Controls.HeaderedItemsControl` расширяет класс `System.Windows.Controls.ItemsControl` заголовком, который используется производными классами для того, чтобы описывать список дочерних элементов.

Свойства класса `System.Windows.Controls.HeaderedItemsControl`:

- **HasHeader** (`System.Boolean`). Получает значение, которое указывает, есть ли у элемента заголовок. `True` если у элемента есть заголовок; иначе — `false`.
- **Header** (`System.Object`). Получает или задает заголовок элемента. Значение по умолчанию — `null`.

## 2. Кисти заднего фона и переднего плана

Элементы управления и некоторые примитивные элементы в WPF обладают рядом свойств, описывающих цвет той или иной их части. Наиболее распространенные из них это свойства, отвечающие за цвет заднего фона (*background*) и переднего плана (*foreground*). В большинстве ситуаций фоном является поверхность элемента управления, а цвет переднего плана используется для окраски текста. Для задания значений свойствам используются разнообразные кисти, которые описываются классами производными от класса `System.Windows.Media.Brush` (рис. 3).

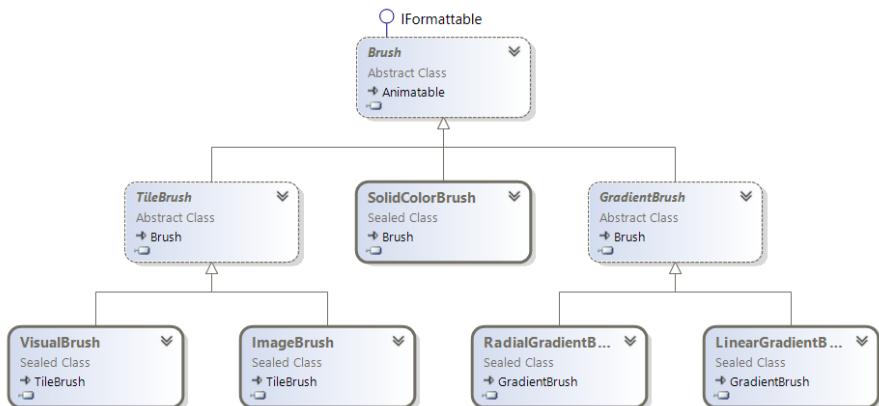


Рис. 3. Диаграмма классов, описывающих кисти

Свойства класса `System.Windows.Media.Brush`:

- **Opacity** (`System.Double`). Получает или задает коэффициент прозрачности кисти. Данное свойство мо-

может принимать значения от 0.0 (прозрачный) до 1.0 (непрозрачный). Значение по умолчанию — 1.0.

Некоторые из кистей требуют указание цвета для разных параметров в виде экземпляра структуры `System.Windows.Media.Color`, который описывает цвет при помощи альфа, красного, зеленого и синего каналов. Существует несколько способов получить такой экземпляр:

- Использовать одно из статических свойств класса `System.Windows.Media.Colors`. Каждое свойство этого класса описывает один из наиболее распространенных цветов. Например, `System.Windows.Media.Colors.Yellow`.
- Использовать один из статических методов структуры `System.Windows.Media.Color`, предназначенных для создания нового экземпляра, например, `System.Windows.Media.Color.FromArgb`.

Когда появляется необходимость установить значение цвета в разметке, то можно указать название одного из свойств класса `System.Windows.Media.Colors` или задать значения для каждого канала явным образом. Для второго варианта допустимы следующие формы записи: "#AARRGGBB", "#ARGB", "#RRGGBB" и "#RGB". В этой записи на месте каждого из символов должна стоять одна любая цифра, записанная в шестнадцатеричной системе счисления.

Свойства структуры `System.Windows.Media.Color`:

- `A` (`System.Byte`). Получает или задает значение альфа канала. Данное свойство может принимать значения от 0 до 255.

- **B** (System.Byte). Получает или задает значение синего канала. Данное свойство может принимать значения от 0 до 255.
- **G** (System.Byte). Получает или задает значение зеленого канала. Данное свойство может принимать значения от 0 до 255.
- **R** (System.Byte). Получает или задает значение красного канала. Данное свойство может принимать значения от 0 до 255.
- **ScA** (System.Single). Получает или задает значение альфа канала. Данное свойство может принимать значения от 0.0 до 1.0.
- **ScB** (System.Single). Получает или задает значение синего канала. Данное свойство может принимать значения от 0.0 до 1.0.
- **ScG** (System.Single). Получает или задает значение зеленого канала. Данное свойство может принимать значения от 0.0 до 1.0.
- **ScR** (System.Single). Получает или задает значение красного канала. Данное свойство может принимать значения от 0.0 до 1.0.

Статические методы структуры System.Windows.Media.Color:

- **FromArgb(a, r, g, b)**. Возвращает цвет на основании указанных значений альфа канала и каналов цветов.
- **FromRgb(r, g, b)**. Возвращает цвет на основании указанных значений каналов цветов.
- **FromScRgb(a, r, g, b)**. Возвращает цвет на основании указанных значений альфа канала и каналов цветов.

## 2.1. Кисть SolidColorBrush

Кисть System.Windows.Media.**SolidColorBrush** используется для заливки сплошным цветом.

Свойства класса System.Windows.Media.**SolidColorBrush**:

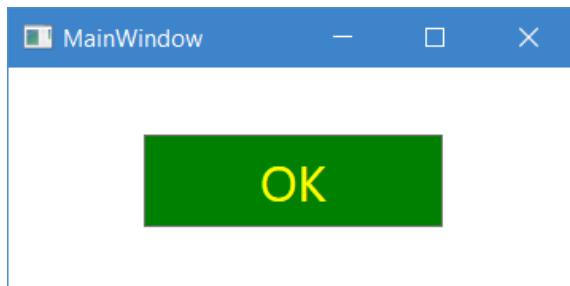
- **Color** (System.Windows.Media.**Color**). Получает или задает цвет кисти. Значение по умолчанию — System.Windows.Media.**Colors.Transparent**.

Приведенный ниже фрагмент разметки демонстрирует кисть (полный пример находится в папке **Wpf.Brushes.SolidColorBrush.Xaml**):

**XAML**

```
<Button Background="Green"
        FontSize="25"
        Foreground="#FFFFFF00"
        Height="46"
        Width="150">
    OK
</Button>
```

Результат приведенной выше разметки показан на рис. 4.



**Рис. 4.** Кисть SolidColorBrush

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Brushes.SolidColorBrush.CSharp**):

C#

```
var button = new Button
{
    Background = Brushes.Green,
    Content = "OK",
    FontSize = 25.0,
    Foreground = new SolidColorBrush(
        Color.FromArgb(a: 255, r: 255, g: 255, b: 0)
    ),
    Height = 46.0,
    Width = 150.0
};
```

## 2.2. Градиентные кисти

Кисти, описывающие градиентную заливку цветом, наследуются от класса `System.Windows.Media.GradientBrush`, который содержит коллекцию градиентных точек, описывающих цвет и координату точки перехода, представленных в виде класса `System.Windows.Media.GradientStop`. Каждый из производных классов интерпретирует данную коллекцию по-своему.

Свойства класса `System.Windows.Media.GradientBrush`:

- **ColorInterpolationMode** (`System.Windows.Media.ColorInterpolationMode`). Получает или задает режим интерполяции цветов градиента. Значение по умолчанию — `System.Windows.Media.ColorInterpolationMode.SRgbLinearInterpolation`.

- **GradientStops** (System.Windows.Media.GradientStopCollection). Получает или задает коллекцию градиентных точек. Значение по умолчанию — пустая коллекция.
- **SpreadMethod** (System.Windows.Media.GradientSpreadMethod). Получает или задает режим распространения градиента, а именно, описание того как необходимо отображать градиент, который начинается или заканчивается в пределах области отображения элемента. Значение по умолчанию — System.Windows.Media.GradientSpreadMethod.Pad.

Для того, чтобы указать режим интерполяции цветов градиента необходимо использовать перечисление System.Windows.Media.ColorInterpolationMode, которое содержит следующие варианты:

- ScRgbLinearInterpolation. Для интерполяции цвета будут использованы свойства System.Windows.Media.Color.ScA, System.Windows.Media.Color.ScR, System.Windows.Media.Color.ScG, System.Windows.Media.Color.ScB.
- SRgbLinearInterpolation. Для интерполяции цвета будут использованы свойства System.Windows.Media.Color.A, System.Windows.Media.Color.R, System.Windows.Media.Color.G, System.Windows.Media.Color.B.

Для того, чтобы указать режим распространения градиента необходимо использовать перечисление System.Windows.Media.GradientSpreadMethod, которое содержит следующие варианты:

- Pad. Значения цвета на концах вектора градиента заполняют оставшееся пространство в своем направлении.

- Reflect. Градиент повторяется в обратном виде.
- Repeat. Градиент повторяется в оригинальном виде. Свойства класса System.Windows.Media.**GradientStop**:
- **Color** (System.Windows.Media.**Color**). Получает или задает цвет градиентной точки. Значение по умолчанию — System.Windows.Media.**Colors.Transparent**.
- **Offset** (System.**Double**). Получает или задает смещение на векторе градиента. Данное свойство может принимать значения в диапазоне от 0.0 (начало вектора) до 1.0 (конец вектора). Значение по умолчанию — 0.0.

При описании точек градиента можно использовать как положительные, так и отрицательные значения, а также значение 0. При этом координата (0, 0) привязана к левому верхнему углу области заливки, а координата (1, 1) к правому нижнему, не зависимо от размеров и пропорций области заливки.

### 2.2.1. Кисть **LinearGradientBrush**

Кисть System.Windows.Media.**LinearGradientBrush** используется для заливки линейным градиентом, который смешивает два и более цветов вдоль прямой линии (вектора градиента).

Свойства класса System.Windows.Media.**LinearGradientBrush**:

- **EndPoint** (System.Windows.**Point**). Получает или задает координаты (в двумерной системе координат), описывающие конец вектора градиента. Значение по умолчанию — точка с координатами (1, 1).

- **StartPoint** (System.Windows.Point). Получает или задает координаты (в двумерной системе координат), описывающие начало вектора градиента. Значение по умолчанию — точка с координатами (0, 0).

Приведенный ниже фрагмент разметки демонстрирует кисть (полный пример находится в папке **Wpf.Brushes.LinearGradientBrush.Xaml**):

### XAML

```
<Button Content="OK" FontSize="25"
       Height="46" Width="150">
    <Button.Background>
        <LinearGradientBrush EndPoint="1,1"
                             StartPoint="0,0">
            <GradientStop Color="Red" Offset="0"/>
            <GradientStop Color="Green"
                           Offset="0.5"/>
            <GradientStop Color="Blue" Offset="1"/>
        </LinearGradientBrush>
    </Button.Background>
    <Button.Foreground>
        <LinearGradientBrush EndPoint="0,1"
                             StartPoint="0,0">
            <GradientStop Color="White" Offset="0"/>
            <GradientStop Color="#FF777777"
                           Offset="0.5"/>
            <GradientStop Color="White" Offset="1"/>
        </LinearGradientBrush>
    </Button.Foreground>
</Button>
```

Результат приведенной выше разметки показан на рис. 5.



**Рис. 5.** Кисть LinearGradientBrush

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Brushes.LinearGradientBrush.CSharp**):

*C#*

```
var backgroundBrush = new LinearGradientBrush
{
    EndPoint = new Point(x: 1.0, y: 1.0),
    StartPoint = new Point(x: 0.0, y: 0.0)
};
backgroundBrush.GradientStops.Add(
    new GradientStop(Colors.Red, offset: 0.0));
backgroundBrush.GradientStops.Add(
    new GradientStop(Colors.Green, offset: 0.5));
backgroundBrush.GradientStops.Add(
    new GradientStop(Colors.Blue, offset: 1.0));
var foregroundBrush = new LinearGradientBrush
{
    EndPoint = new Point(x: 0.0, y: 1.0),
    StartPoint = new Point(x: 0.0, y: 0.0)
};
foregroundBrush.GradientStops.Add(
    new GradientStop(Colors.White, offset: 0.0));
foregroundBrush.GradientStops.Add(
    new GradientStop(
        Color.FromArgb(a: 0xFF, r: 0x77, g: 0x77, b: 0x77),
        offset: 0.5));
```

```
foregroundBrush.GradientStops.Add(  
    new GradientStop(Colors.White, offset: 1.0));  
  
var button = new Button  
{  
    Background = backgroundBrush,  
    Content = "OK",  
    FontSize = 25.0,  
    Foreground = foregroundBrush,  
    Height = 46.0,  
    Width = 150.0  
};
```

### 2.2.2. Кисть *RadialGradientBrush*

Кисть [System.Windows.Media.RadialGradientBrush](#) используется для заливки радиальным (круговым) градиентом, который смешивает два и более цветов вдоль радиуса окружности.

Свойства класса [System.Windows.Media.RadialGradientBrush](#):

- **Center** ([System.Windows.Point](#)). Получает или задает координаты (в двумерной системе координат) центра окружности градиента.
- **GradientOrigin** ([System.Windows.Point](#)). Получает или задает координаты (в двумерной системе координат) фокусной точки, которая определяет начало градиента. Значение по умолчанию — точка с координатами (0.5, 0.5).
- **RadiusX** ([System.Double](#)). Получает или задает горизонтальный радиус окружности градиента. Значение по умолчанию — 0.5.

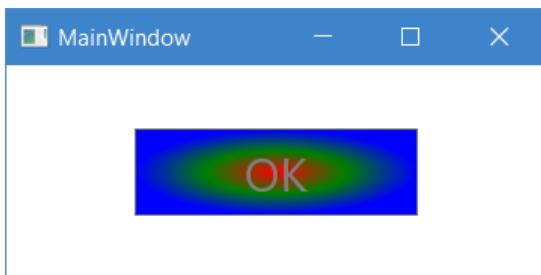
- **RadiusY** (`System.Double`). Получает или задает вертикальный радиус окружности градиента. Значение по умолчанию — 0.5.

Приведенный ниже фрагмент разметки демонстрирует кисть (полный пример находится в папке `Wpf.Brushes.RadialGradientBrush.Xaml`):

### XAML

```
<Button Content="OK" FontSize="25" Foreground="Gray"
       Height="46" Width="150">
    <Button.Background>
        <RadialGradientBrush GradientOrigin="0.5,0.5">
            <GradientStop Color="Red" Offset="0"/>
            <GradientStop Color="Green"
                           Offset="0.5"/>
            <GradientStop Color="Blue" Offset="1"/>
        </RadialGradientBrush>
    </Button.Background>
</Button>
```

Результат приведенной разметки показан на рис. 6.



**Рис. 6.** Кисть `RadialGradientBrush`

Рассмотренной выше разметке соответствует следующий код (полный пример находится в `Wpf.Brushes.RadialGradientBrush.CSharp`):

## C#

```

var backgroundBrush = new RadialGradientBrush {
    GradientOrigin = new Point(0.5, 0.5) };

backgroundBrush.GradientStops.Add(
    new GradientStop(Colors.Red, offset: 0.0));

backgroundBrush.GradientStops.Add(
    new GradientStop(Colors.Green, offset: 0.5));

backgroundBrush.GradientStops.Add(
    new GradientStop(Colors.Blue, offset: 1.0));

var button = new Button
{
    Background = backgroundBrush,
    Content = "OK",
    FontSize = 25.0,
    Foreground = Brushes.Gray,
    Height = 46.0,
    Width = 150.0
};

```

## 2.3. Плиточные кисти

Кисти, позволяющие использовать изображения для заливки, наследуются от класса `System.Windows.Media.TileBrush`, который описывает принцип плиточной заливки.

Свойства класса `System.Windows.Media.TileBrush`:

- **AlignmentX** (`System.Windows.Media.AlignmentX`). Получает или задает горизонтальное выравнивание для содержимого базовой плитки. Значение по умолчанию — `System.Windows.Media.AlignmentX.Center`.
- **AlignmentY** (`System.Windows.Media.AlignmentY`). Получает или задает вертикальное выравнивание для

содержимого базовой плитки. Значение по умолчанию — System.Windows.Media.AlignmentY.Center.

- **Stretch** (System.Windows.Media.Stretch). Получает или задает режим растяжения содержимого. Значение по умолчанию — System.Windows.Media.Stretch.Fill.
- **TileMode** (System.Windows.Media.TileMode). Получает или задает режим плиточного заполнения, которое используется, если базовая плитка меньше области заливки. Значение по умолчанию — System.Windows.Media.TileMode.None.

Для того, чтобы указать горизонтальное выравнивание для содержимого базовой плитки необходимо использовать перечисление System.Windows.Media.AlignmentX, которое содержит следующие варианты:

- Center. Содержимое центрируется.
- Left. Содержимое выравнивается по левому краю.
- Right. Содержимое выравнивается по правому краю.

Для того, чтобы указать вертикальное выравнивание для содержимого базовой плитки необходимо использовать перечисление System.Windows.Media.AlignmentY, которое содержит следующие варианты:

- Bottom. Содержимое выравнивается по нижнему краю.
- Center. Содержимое центрируется.
- Top. Содержимое выравнивается по верхнему краю.

Для того, чтобы указать режим растяжения содержимого необходимо использовать перечисление System.Windows.Media.Stretch, которое содержит следующие варианты:

- Fill. Содержимое растягивается, чтобы заполнить все доступное пространство. Пропорции не сохраняются.
- None. Нет растяжения. Содержимое сохраняет оригинальные размеры.
- Uniform. Содержимое растягивается, чтобы заполнить все доступное пространство. Пропорции сохраняются. При этом содержимое будет показано целиком.
- UniformToFill. Содержимое растягивается, чтобы заполнить все доступное пространство. Пропорции сохраняются. При этом если содержимое целиком не помещается, то оно будет обрезано.

Для того, чтобы указать режим плиточного заполнения необходимо использовать перечисление `System.Windows.Media.TileMode`, которое содержит следующие варианты:

- FlipX. То же самое, что и `System.Windows.Media.TileMode.Tile`, но чередующиеся колонки отображаются зеркально. Базовая плитка отображается в оригинальном виде.
- FlipXY. Комбинация `System.Windows.Media.TileMode.FlipX` и `System.Windows.Media.TileMode.FlipY`.
- FlipY. То же самое, что и `System.Windows.Media.TileMode.Tile`, но чередующиеся строки отображаются зеркально. Базовая плитка отображается в оригинальном виде.
- None. Базовая плитка отображается без повторений, а оставшееся пространство остается прозрачным.
- Tile. Базовая плитка отображается, а оставшееся пространство заполняется повторением базовой плитки. Правый край одной плитки стыкуется с левым другой, а верхний край стыкуется с нижним.

### 2.3.1. Кисть *ImageBrush*

Кисть `System.Windows.Media.ImageBrush` используется для плиточной заливки изображением.

Свойства класса `System.Windows.Media.ImageBrush`:

- **ImageSource** (`System.Windows.Media.ImageSource`). Получает или задает источник изображения.

Приведенный ниже фрагмент разметки демонстрирует кисть (полный пример находится в папке **Wpf.Brushes.ImageBrush.Xaml**):

#### XAML

```
<Button Content="OK" FontSize="25" Foreground="Gray"
       Height="46" Width="150">
    <Button.Background>
        <ImageBrush ImageSource="/Succulent.jpg"
                    Stretch="UniformToFill"/>
    </Button.Background>
</Button>
```

Результат приведенной выше разметки показан на рис. 7.

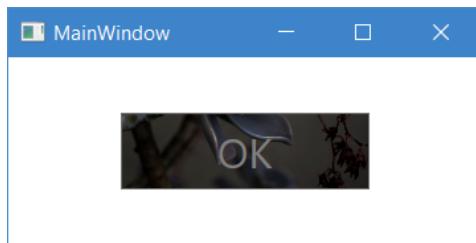


Рис. 7. Кисть *ImageBrush*

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Brushes.ImageBrush.CSharp**):

**C#**

```

var backgroundBrush = new ImageBrush
{
    ImageSource = new BitmapImage(new Uri("Succulent.
        jpg", UriKind.Relative)),
    Stretch = Stretch.UniformToFill
};
var button = new Button
{
    Background = backgroundBrush,
    Content = "OK",
    FontSize = 25.0,
    Foreground = Brushes.Gray,
    Height = 46.0,
    Width = 150.0
};

```

**2.3.2. Кисть VisualBrush**

Кисть System.Windows.Media.VisualBrush используется для заливки какого-либо визуального объекта.

Свойства класса System.Windows.Media.VisualBrush:

- **Visual** (System.Windows.Media.Visual). Получает или задает визуальный элемент, внешний вид которого представляет из себя содержимое для заливки. Значение по умолчанию — **null**.

Приведенный ниже фрагмент разметки демонстрирует кисть (полный пример находится в папке **Wpf.Brushes.VisualBrush.Xaml**):

**XAML**

```

<StackPanel HorizontalAlignment="Center"
    VerticalAlignment="Center">

```

```
<TextBox x:Name="visual" FontSize="25"
         Text="123" Width="150"/>
<TextBlock>
    <TextBlock.Background>
        <VisualBrush Visual=
            "{Binding ElementName=visual}"/>
    </TextBlock.Background>
</TextBlock>
</StackPanel>
```

Результат приведенной выше разметки показан на рис. 8.

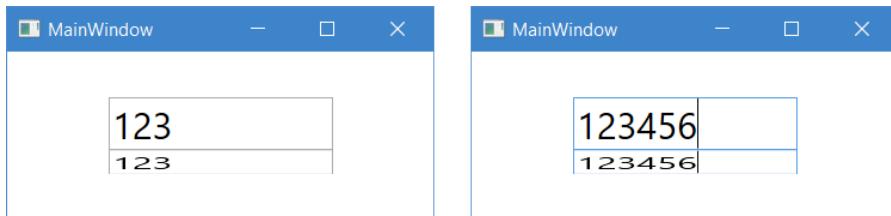


Рис. 8. Кисть VisualBrush

В примере было использовано расширение разметки для того, чтобы указать какой именно элемент является содержимым: `Visual="{Binding ElementName=visual}"`. Это расширение разметки необходимо для описания привязки данных. Подробно принцип его функционирования будет рассмотрен в будущих разделах.

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Brushes.VisualBrush.CSharp**):

C#

```
var textBox = new TextBox { FontSize = 25.0,
                           Text = "123", Width = 150.0 };
```

## 2. Кисти заднего фона и переднего плана

```
var textBlock = new TextBlock { Background =
    new VisualBrush(visual: textBox) };

var stackPanel = new StackPanel
{
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center
};
stackPanel.Children.Add(textBox);
stackPanel.Children.Add(textBlock);
```

# 3. Шрифты

Элементы управления обладают набором свойств, отвечающих за шрифт, используемый при отображении текста.

## 3.1. Размер шрифта

Свойство, отвечающее за размер шрифта (`FontSize`) может принимать только положительные значения. Указанное значение измеряется в аппаратно-независимых единицах, что является отклонением от общепринятых единиц измерения для описания размера шрифта — пунктов. Для того, чтобы преобразовать единицы, используемые в WPF в пункты, необходимо их умножить на 0.75, т. е. 36 пунктов эквивалентно 48 единицам в WPF.

## 3.2. Раствжение или сжатие текста

Для того, чтобы указать режим растяжения или сжатия текста (`FontStretch`) необходимо использовать статические свойства класса `System.Windows.FontStretches`, каждое из которых указывает на сколько процентов, относительно нормального состояния, необходимо растянуть или сжать текст.

FontStretch	% от нормального
UltraCondensed	50,0%
ExtraCondensed	62,5%
Condensed	75,0%
SemiCondensed	87,5%
Normal	100,0%

FontStretch	% от нормального
Medium	100,0%
SemiExpanded	112,5%
Expanded	125,0%
ExtraExpanded	150,0%
UltraExpanded	200,0%

### 3.3. Плотность текста

Для того, чтобы указать плотность текста (FontWeight) необходимо использовать статические свойства класса System.Windows.FontWeights, каждое из которых указывает значение в диапазоне от 1 до 999. Чем меньше значение, тем меньше плотность текста, и наоборот, чем больше значение, тем больше плотность текста.

FontWeight	Значение
Thin	100
ExtraLight UltraLight	200
Light	300
Normal Regular	400
Medium	500
DemiBold SemiBold	600
Bold	700
ExtraBold UltraBold	800
Black Heavy	900
ExtraBlack ExtraHeavy	950

### 3.4. Наклон текста

Для того, чтобы указать режим наклона текста (`FontStyle`) необходимо использовать статические свойства класса `System.Windows.FontStyles`.

FontStyle	Описание
Normal	Наклон текста отсутствует.
Italic	Используется курсивный вариант отображения символов выбранного шрифта.
Oblique	Курсивный вариант шрифта генерируется автоматически на основании трансформации обычного. Используется в тех случаях, когда шрифт не обладает изначально курсивным вариантом.

### 3.5. Семейство шрифтов

Семейство шрифтов (`FontFamily`) определяется набором родственных гарнитур, типографические правила и внешний вид символов, для которых определяются отдельно друг от друга. При этом расцениваются они как составные части одного семейства. Например, **Arial Regular**, **Arial Bold**, **Arial Italic** и **Arial Bold Italic** являются составными частями семейства шрифтов **Arial**.

Если указать, что элемент должен использовать семейство шрифтов **Arial**, и при этом указать плотность текст `System.Windows.FontWeights.Bold`, то автоматически будет выбрана гарнитура **Arial Bold**.

В приведенной ниже разметке создаются две кнопки, настройки шрифтов которых являются идентичными с точки зрения получившегося результата:

## XAML

```
<Button FontFamily="Arial" FontWeight="Bold">Button  
Text</Button>  
<Button FontFamily="Arial Bold">Button Text</Button>
```

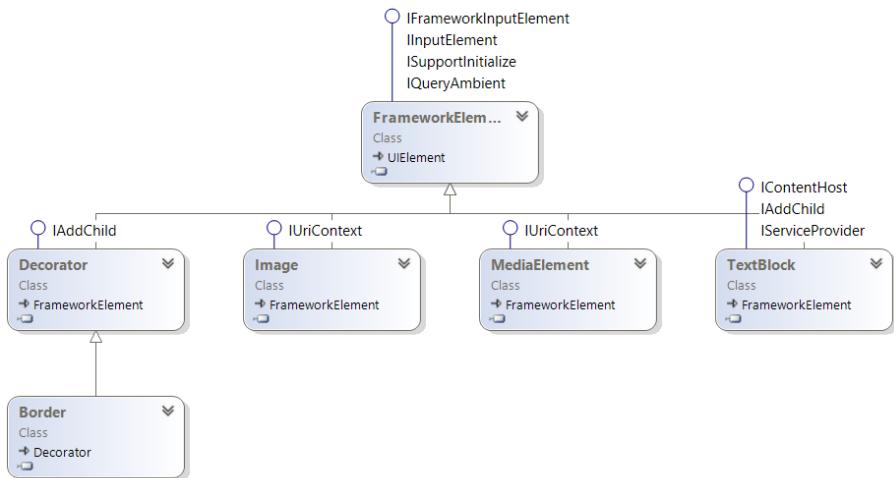
### 3.6. Наследование шрифтов

Если задать свойство, отвечающее за какую-то характеристику шрифта, то значение этого свойства будет наследоваться всеми дочерними элементами, до тех пор, пока какой-то из них не переопределит его явным образом. Другими словами, если задать семейство шрифтов окну верхнего уровня (элемент `<Window>`), то это же семейство шрифтов будет использовано для всех элементов окна.

Наследование данных свойств может проходить даже через элементы, которые не поддерживают его, например, элемент `<StackPanel>`.

## 4. Примитивные элементы

WPF предоставляет большой выбор элементов управления для построения пользовательского интерфейса. Однако, как было сказано ранее, далеко не все элементы в WPF являются элементами управления. К ним относятся те элементы, которые обладают определенным предустановленным поведением и возможностью взаимодействия с пользователем, а также, шаблоном внешнего вида. Существует ряд, так называемых, примитивных элементов (рис. 9), которые предоставляют какую-то очень базовую функциональность (вывод текста, изображения, видео и т.д.).



**Рис. 9.** Диаграмма классов, описывающих примитивные элементы

## 4.1. Элемент **TextBlock**

Элемент `System.Windows.Controls.TextBlock` представляет из себя легковесную версию элемента управления, который позволяет отображать текст.

Свойства класса `System.Windows.Controls.TextBlock`:

- **Background** (`System.Windows.Media.Brush`). Получает или задает кисть, которая используется для заливки заднего фона элемента. Значение по умолчанию — `null`.
- **BaselineOffset** (`System.Double`). Получает или задает смещение от базовой линии текста. Данное свойство может принять значение `System.Double.NaN`, которое означает, что смещение должно быть вычислено как оптимальное на основании текущих настроек шрифта. Значение по умолчанию — `System.Double.NaN`.
- **FontFamily** (`System.Windows.Media.FontFamily`). Получает или задает семейство шрифтов. Значение по умолчанию — `System.Windows.SystemFonts.MessageFontFamily`.
- **FontSize** (`System.Double`). Получает или задает размер шрифта. Данное свойство может принимать только положительные значения. Значение по умолчанию — `System.Windows.SystemFonts.MessageFontSize`.
- **FontStretch** (`System.Windows.FontStretch`). Получает или задает режим растяжения или сжатия шрифта. Значение по умолчанию — `System.Windows.FontStretches.Normal`.
- **FontStyle** (`System.Windows.FontStyle`). Получает или задает режим наклона текста. Значение по умолчанию — `System.Windows.SystemFonts.MessageFontSize`.

- **FontWeight** (System.Windows.FontWeight). Получает или задает плотность текста. Значение по умолчанию — System.Windows.SystemFonts.MessageFontWeight.
- **Foreground** (System.Windows.Media.Brush). Получает или задает кисть, которая используется для заливки переднего плана элемента (текста). Значение по умолчанию — System.Windows.Media.Brushes.Black.
- **IsHyphenationEnabled** (System.Boolean). Получает или задает значение, которое указывает, включен ли перенос слов с использование дефисов. **True** если перенос слов с использование дефисов включен; иначе — **false**. Значение по умолчанию — **false**.
- **LineHeight** (System.Double). Получает или задает высоту строки. Данное свойство может принять значение System.Double.NaN, которое означает, что высота строки должна быть вычислена как оптимальная на основании текущих настроек шрифта. Значение по умолчанию — System.Double.NaN.
- **Text** (System.String). Получает или задает текст. Значение по умолчанию — System.String.Empty.
- **TextAlignment** (System.Windows.TextAlignment). Получает или задает режим горизонтального выравнивания для текста. Значение по умолчанию — System.Windows.TextAlignment.Left.
- **TextTrimming** (System.Windows.TextTrimming). Получает или задает режим обрезания содержимого при переполнении. Значение по умолчанию — System.Windows.TextTrimming.None.

- **TextWrapping** (System.Windows.TextWrapping). Получает или задает режим переноса текста. Значение по умолчанию — System.Windows.TextWrapping.NoWrap.

Для того, чтобы указать режим обрезания содержимого при переполнении необходимо использовать перечисление System.Windows.Controls.TextTrimming, которое содержит следующие варианты:

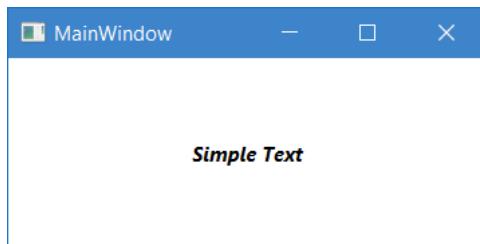
- CharacterEllipsis. Текст обрезается на границе символа. Многоточие (...) помещается на место обрезанного текста.
- None. Текст не обрезается.
- WordEllipsis. Текст обрезается на границе слова. Многоточие (...) помещается на место обрезанного текста.

Приведенный ниже фрагмент разметки демонстрирует элемент (полный пример находится в папке **Wpf.Primitives.TextBlock.Xaml**):

### XAML

```
<TextBlock FontStyle="Italic" FontWeight="Bold"
           Text="Simple Text"/>
```

Результат приведенной выше разметки показан на рис. 10.



**Рис. 10.** Примитивный элемент TextBlock

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Primitives.TextBlock.CSharp**):

C#

```
var textBlock = new TextBlock
{
    FontStyle = FontStyles.Italic,
    FontWeight = FontWeights.Bold,
    Text = "Simple Text"
};
```

## 4.2. Элемент Image

Элемент `System.Windows.Controls.Image` предназначен для вывода изображений.

Свойства класса `System.Windows.Controls.Image`:

- **Source** (`System.Windows.Media.ImageSource`). Получает или задает источник изображения. Значение по умолчанию — `null`.
- **Stretch** (`System.Windows.Media.Stretch`). Получает или задает режим растяжения изображения. Значение по умолчанию — `System.Windows.Media.Stretch.Uniform`.
- **StretchDirection** (`System.Windows.Controls.StretchDirection`). Получает или задает режим изменения размеров изображения. Значение по умолчанию — `System.Windows.Controls.StretchDirection.Both`.

События класса `System.Windows.Controls.Image`:

- **ImageFailed** (`System.EventHandler<System.Windows.ExceptionRoutedEventArgs>`). Срабатывает, когда происходит ошибка при считывании источника изображения.

Для того, чтобы указать режим изменения размеров изображения необходимо использовать перечисление System.Windows.Media.StretchDirection, которое содержит следующие варианты:

- Both. Изображение растягивается в любом направлении с учетом значения свойства System.Windows.Controls.Image.Stretch.
- DownOnly. Изображение может только уменьшаться в случае, если оно больше, чем доступное пространство родительского элемента.
- UpOnly. Изображение может только растягиваться в случае, если оно меньше, чем доступное пространство родительского элемента.

Приведенный ниже фрагмент разметки демонстрирует элемент (полный пример находится в папке **Wpf.Primitives.Image.Xaml**):

#### XAML

```
<Image Margin="10" Source="Succulent.jpg"/>
```

Результат приведенной выше разметки показан на рис. 11.

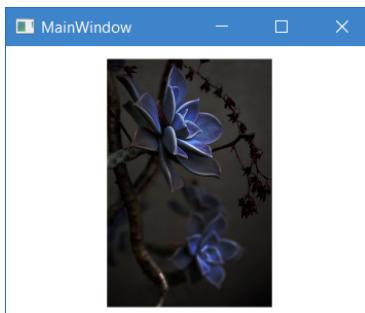


Рис. 11. Примитивный элемент Image.

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Primitives.Image.CSharp**):

C#

```
var image = new Image
{
    Margin = new Thickness(10.0),
    Source = new BitmapImage(new Uri("/Succulent.jpg",
        UriKind.Relative))
};
```

### 4.3. Элемент MediaElement

Элемент **System.Windows.Controls.MediaElement** предназначен для проигрывания аудио или видео содержимого.

Свойства класса **System.Windows.Controls.MediaElement**:

- **Balance** (**System.Double**). Получает или задает соотношение распределения громкости между динамиками. Данное свойство может принимать значения от -1.0 (вся громкость на левый динамик) до 1.0 (вся громкость на правый динамик). Значение по умолчанию — 0.0.
- **BufferingProgress** (**System.Double**). Получает прогресс буферизации содержимого. Данное свойство возвращает значение в диапазоне от 0.0 (прогресс 0%) до 1.0 (прогресс 100%).
- **CanPause** (**System.Boolean**). Получает значение, которое указывает, может ли быть проигрывание содержимого поставлено на паузу. **True** если пауза допустима; иначе — **false**.

- **DownloadProgress** (System.Double). Получает прогресс скачивания содержимого, расположенного на удаленном сервере. Данное свойство возвращает значение в диапазоне от 0.0 (прогресс 0%) до 1.0 (прогресс 100%).
- **HasAudio** (System.Boolean). Получает значение, которое указывает, обладает ли содержимое звуком. **True** если содержимое обладает звуком; иначе — **false**.
- **HasVideo** (System.Boolean). Получает значение, которое указывает, обладает ли содержимое видео. **True** если содержимое обладает видео; иначе — **false**.
- **IsBuffering** (System.Boolean). Получает значение, которое указывает, буферизируется ли содержимое. **True** если содержимое буферизируется; иначе — **false**.
- **IsMuted** (System.Boolean). Получает или задает значение, которое указывает, отключен ли звук для содержимого. **True** если звук отключен; иначе — **false**. Значение по умолчанию — **false**.
- **LoadedBehavior** (System.Windows.Controls.MediaState). Получает или задает поведение для содержимого после его загрузки. Значение по умолчанию — System.Windows.Controls.MediaState.Play.
- **NaturalDuration** (System.Windows.Duration). Получает действительную продолжительность содержимого.
- **NaturalVideoHeight** (System.Int32). Получает действительную высоту видео.
- **NaturalVideoWidth** (System.Int32). Получает действительную ширину видео.
- **ScrubbingEnabled** (System.Boolean). Получает или задает значение, которое указывает будет ли элемент

обновлять кадры при проматывании, в состоянии паузы. **True** если необходимо обновлять кадры; иначе — **false**. Значение по умолчанию — **false**.

- **Source** ([System.Uri](#)). Получает или задает источник содержимого. Значение по умолчанию — **null**.
- **SpeedRatio** ([System.Double](#)). Получает или задает коэффициент скорости проигрывания содержимого. Данное свойство может принимать значения в диапазоне от 0.0 до [System.Double.PositiveInfinity](#). Отрицательные значения преобразовываются в 0.0. Значения в диапазоне от 0.0 до 1.0 замедляют проигрывание, а значения больше 1.0 его ускоряют. Значение по умолчанию — 1.0.
- **Stretch** ([System.Windows.Media.Stretch](#)). Получает или задает режим растяжения содержимого. Значение по умолчанию — [System.Windows.Media.Stretch.Uniform](#).
- **StretchDirection** ([System.Windows.Controls.StretchDirection](#)). Получает или задает режим изменения размеров содержимого. Значение по умолчанию — [System.Windows.Controls.StretchDirection.Both](#).
- **Position** ([System.TimeSpan](#)). Получает или задает текущую позицию проигрывания для содержимого. Значение по умолчанию — [System.TimeSpan.Zero](#).
- **UnloadedBehavior** ([System.Windows.Controls.MediaState](#)). Получает или задает поведение для содержимого после его выгрузки.
- **Volume** ([System.Double](#)). Получает или задает уровень громкости. Данное свойство может принимать значения в диапазоне от 0.0 (звук отключен) до 1.0 (максимальная громкость). Значение по умолчанию — 0.5.

События класса System.Windows.Controls.MediaElement:

- **BufferingEnded** (System.Windows.RoutedEventHandler). Срабатывает, когда буферизация содержимого завершается.
- **BufferingStarted** (System.Windows.RoutedEventHandler). Срабатывает, когда буферизация содержимого начинается.
- **MediaEnded** (System.Windows.RoutedEventHandler). Срабатывает, когда проигрывание содержимого завершается.
- **MediaFailed** (System.EventHandler<System.Windows.ExceptionRoutedEventArgs>). Срабатывает, когда происходит ошибка при считывании содержимого.
- **MediaOpened** (System.Windows.RoutedEventHandler). Срабатывает, когда загрузка содержимого завершается.

Методы класса System.Windows.Controls.MediaElement:

- **Close()**. Закрывает содержимое.
- **Pause()**. Ставит проигрывание содержимого на паузу.
- **Play()**. Начинает проигрывание содержимого.
- **Stop()**. Останавливает проигрывание содержимого.

Для того, чтобы указать поведение содержимого при загрузке или выгрузке необходимо использовать перечисление System.Windows.Controls.MediaState, которое содержит следующие варианты:

- Close. Содержимое необходимо закрыть и высвободить все ресурсы, занимаемые им.

- Manual. Ручное управления проигрыванием содержимого. Содержимое будет загружено, но проигрывание не будет автоматически запущено.
- Pause. Содержимое необходимо поставить на паузу.
- Play. Будет запущено проигрывание содержимого.
- Stop. Содержимое будет остановлено. При этом ресурсы, занимаемые им, не будут высвобождены.

Приведенный ниже фрагмент разметки демонстрирует элемент (полный пример находится в папке **Wpf.Primitives.MediaElement.Xaml**):

### XAML

```
<MediaElement Source="KellyMcGarry.mp4"/>
```

Результат приведенной выше разметки показан на рис. 12.



Рис. 12. Примитивный элемент MediaElement

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Primitives.MediaElement.CSharp**):

## C#

```
var mediaElement = new MediaElement
{
    Source = new Uri("KellyMcGarry.mp4",
                      UriKind.Relative)
};
```

## 4.4. Элемент Border

Элемент System.Windows.Controls.Border предназначен для вывода заднего фона, рамки или того и другого.

Свойства класса System.Windows.Controls.Border:

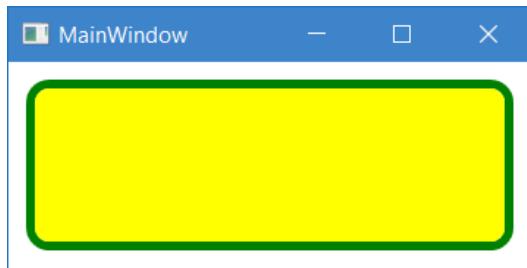
- **Background** (System.Windows.Media.Brush). Получает или задает кисть, которая используется для заливки области между границами элемента.
- **BorderBrush** (System.Windows.Media.Brush). Получает или задает кисть, которая используется для заливки рамки элемента.
- **BorderThickness** (System.Windiws.Thickness). Получает или задает толщину рамки.
- **CornerRadius** (System.Windows.CornerRadius). Получает или задает радиус округления каждого из углов рамки.
- **Padding** (System.Windows.Thickness). Получает или задает внутренние отступы для элемента (между рамкой и содержимым).

Приведенный ниже фрагмент разметки демонстрирует элемент (полный пример находится в папке Wpf.Primitives.Border.Xaml):

## XAML

```
<Border Background="Yellow"
        BorderBrush="Green"
        BorderThickness="5"
        CornerRadius="10"
        Margin="10"/>
```

Результат приведенной выше разметки показан на рис. 13.



**Рис. 13.** Примитивный элемент Border

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Primitives.Border.CSharp**):

C#

```
var border = new Border
{
    Background = Brushes.Yellow,
    BorderBrush = Brushes.Green,
    BorderThickness = new Thickness(5.0),
    CornerRadius = new CornerRadius(10.0),
    Margin = new Thickness(10.0)
};
```

# 5. Элемент управления Label

Элемент управления `System.Windows.Controls.Label` необходим для того, чтобы отобразить текст, связанный с другим элементом управления. Данный элемент управления наследуется от класса `System.Windows.Controls.ContentControl` (рис. 14) и тем самым поддерживает описанную ранее модель управления содержимым, т.е. может содержать не только текст, а фактически, все, что угодно. Несмотря на это чаще всего его применяют именно для отображения текста из-за его отличительной особенности, которая отсутствует у рассмотренного ранее примитивного элемента `System.Windows.Controls.TextBlock`. Это поддержка мнемонических команд — нажатия клавиш быстрого доступа, для передачи фокуса связанному с ним элементу управления.

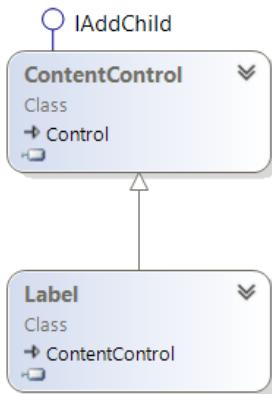


Рис. 14. Диаграмма классов, описывающих метки

Свойства класса System.Windows.Controls.Label:

- **Target** (System.Windows.UIElement). Получает или задает элемент, который должен получить фокус при нажатии клавиши управления, связанной с меткой. Значение по умолчанию — **null**.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке **Wpf.Controls.Label.Xaml**):

### XAML

```
<Grid HorizontalAlignment="Center"
      VerticalAlignment="Center">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Label Grid.Column="0"
          Target="{Binding ElementName=username}">
        Username:
    </Label>
    <TextBox x:Name="username" Grid.Column="1"
             Width="200"/>
</Grid>
```

Результат приведенной выше разметки показан на рис. 15.



Рис. 15. Элемент управления Label

Для того, чтобы указать клавишу быстрого доступа, необходимо поставить символ подчеркивания (\_) перед символом необходимой клавиши. Если возникает необходимость вывести сам символ подчеркивания, то его необходимо экранировать еще одним таким же символом (\_).

Для того, чтобы указать элемент, который должен получить фокус при нажатии клавиши быстрого доступа, необходимо использовать свойство System.Windows.Controls.Label.Target.

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.Label.CSharp**):

C#

```
var textBox = new TextBox { Width = 200.0 };
var label = new Label { Content = "_Username:",
                      Target = textBox };

var grid = new Grid
{
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center
};

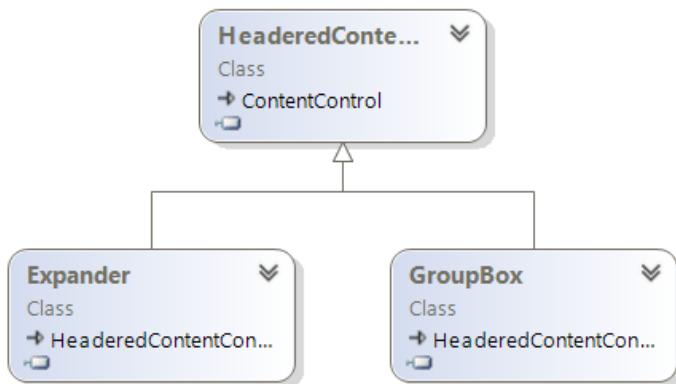
grid.ColumnDefinitions.Add(new ColumnDefinition
    { Width = GridLength.Auto });
grid.ColumnDefinitions.Add(new ColumnDefinition());

grid.Children.Add(label);
grid.Children.Add(textBox);

Grid.SetColumn(label, 0);
Grid.SetColumn(textBox, 1);
```

## 6. Группирующие элементы управления

Группирующие элементы управления в WPF являются наследниками класса `System.Windows.Controls.HeaderedContentControl` (рис. 16). Они предназначены для того, чтобы сгруппировать несколько логически связанных элементов управления вместе и дать получившейся группе название.



**Рис. 16.** Диаграмма классов, описывающих группирующие элементы

### 6.1. Элемент управления `GroupBox`

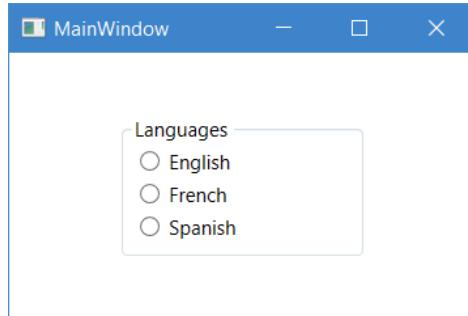
Элемент управления `System.Windows.Controls.GroupBox` позволяет «обернуть» содержимое в рамку с заголовком.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке `Wpf.Controls.GroupBox.Xaml`):

**XAML**

```
<GroupBox Header="Languages"
          HorizontalAlignment="Center"
          VerticalAlignment="Center"
          Width="150">
    <StackPanel Margin="5">
        <RadioButton>English</RadioButton>
        <RadioButton Margin="0,5,0,0">French
                    </RadioButton>
        <RadioButton Margin="0,5,0,0">Spanish
                    </RadioButton>
    </StackPanel>
</GroupBox>
```

Результат приведенной выше разметки показан на рис. 17.



**Рис. 17.** Элемент управления **GroupBox**

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.GroupBox.CSharp**):

**C#**

```
var radioButton1 = new RadioButton { Content =
    "English" };
```

```
var radioButton2 = new RadioButton
{
    Content = "French",
    Margin = new Thickness(0.0, 5.0, 0.0, 0.0)
};
var radioButton3 = new RadioButton
{
    Content = "Spanish",
    Margin = new Thickness(0.0, 5.0, 0.0, 0.0)
};

var stackPanel = new StackPanel { Margin =
    new Thickness(5.0) };
stackPanel.Children.Add(radioButton1);
stackPanel.Children.Add(radioButton2);
stackPanel.Children.Add(radioButton3);

var groupBox = new GroupBox
{
    Content = stackPanel,
    Header = "Languages",
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center,
    Width = 150.0
};
```

## 6.2. Элемент управления Expander

Элемент управления System.Windows.Controls.Expander позволяет задать заголовок своему содержимому, помещенному в область, которую можно при необходимости скрывать или отображать.

Свойства класса System.Windows.Controls.Expander:

- **ExpandDirection** (System.Windows.Controls.ExpandDirection). Получает или задает направления выпа-

дения содержимого. Значение по умолчанию — System.Windows.Controls.ExpandDirection.Down.

- **IsExpanded** (System.Boolean). Получает или задает значение, которое указывает, отображается ли содержимое элемента. **True** если содержимое отображается; иначе — **false**. Значение по умолчанию — **false**.
- События класса System.Windows.Controls.Expander:
- **Collapsed** (System.Windows.RoutedEventHandler). Срабатывает, когда содержимое группирующего элемента скрывается.
  - **Expanded** (System.Windows.RoutedEventHandler). Срабатывает, когда содержимое группирующего элемента отображается.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке Wpf.Controls.Expander.Xaml):

### XAML

```
<Expander Header="Languages" Margin="30"
           Width="200">
    <StackPanel Margin="5">
        <RadioButton>English</RadioButton>
        <RadioButton Margin="0,5,0,0">French
                     </RadioButton>
        <RadioButton Margin="0,5,0,0">Spanish
                     </RadioButton>
    </StackPanel>
</Expander>
```

Результат приведенной выше разметки показан на рис. 18.

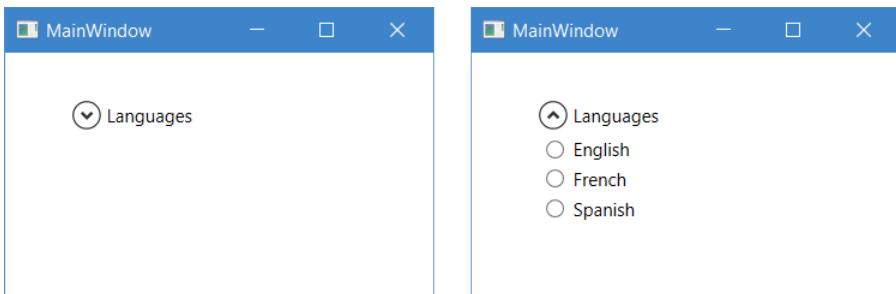


Рис. 18. Элемент управления Expander

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.Expander.CSharp**):

C#

```
var radioButton1 = new RadioButton { Content =
    "English" };
var radioButton2 = new RadioButton
{
    Content = "French",
    Margin = new Thickness(0.0, 5.0, 0.0, 0.0)
};
var radioButton3 = new RadioButton
{
    Content = "Spanish",
    Margin = new Thickness(0.0, 5.0, 0.0, 0.0)
};

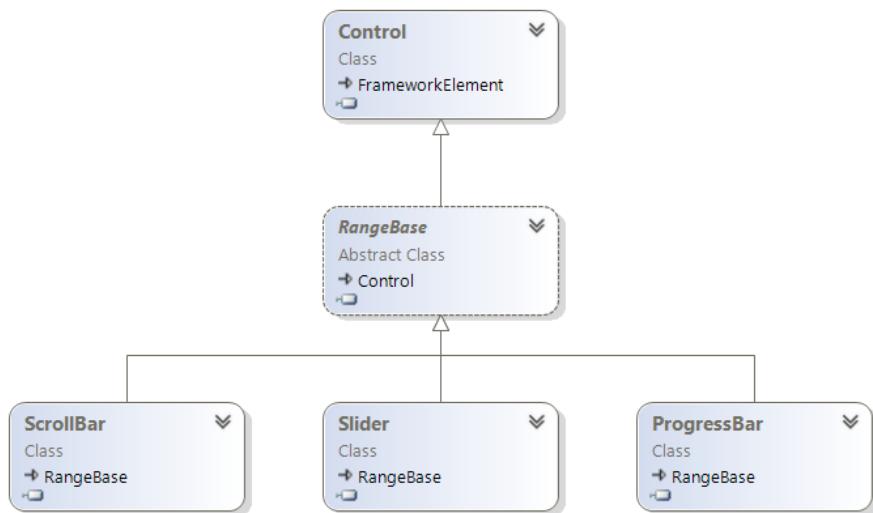
var stackPanel = new StackPanel { Margin =
    new Thickness(5.0) };
stackPanel.Children.Add(radioButton1);
stackPanel.Children.Add(radioButton2);
stackPanel.Children.Add(radioButton3);

var expander = new Expander
{
```

```
Content = stackPanel,  
Header = "Languages",  
Margin = new Thickness(30.0),  
Width = 200.0  
};
```

## 7. Диапазонные элементы управления

Выбор определенного значения из допустимого диапазона значений является довольно распространенной задачей при взаимодействии с приложением. Элементы управления, которым необходима подобная функциональность, наследуются от базового класса `System.Windows.Controls.Primitives.RangeBase` (рис. 19).



**Рис. 19.** Диаграмма классов, описывающих диапазонные элементы управления

Свойства класса `System.Windows.Controls.Primitives.RangeBase`:

- **LargeChange** (`System.Double`). Получает или задает значение, которое необходимо отнимать или добав-

лять к текущему значению свойства System.Windows.Controls.Primitives.RangeBase.Value. Значение по умолчанию — 1.0.

- **Maximum** (System.Double). Получает или задает максимальное значение для свойства System.Windows.Controls.Primitives.RangeBase.Value. Значение по умолчанию — 1.0.
- **Minimum** (System.Double). Получает или задает минимальное значение для свойства System.Windows.Controls.Primitives.RangeBase.Value. Значение по умолчанию — 0.0.
- **SmallChange** (System.Double). Получает или задает значение, которое необходимо отнимать или добавлять к текущему значению свойства System.Windows.Controls.Primitives.RangeBase.Value. Значение по умолчанию — 0.1.
- **Value** (System.Double). Получает или задает текущее значение, установленное в элементе. Значение по умолчанию — 0.0.

События класса System.Windows.Controls.Primitives.RangeBase:

- **ValueChanged** (System.Windows.RoutedPropertyChangedEventHandler<System.Double>). Срабатывает, когда изменяется значение свойства System.Windows.Controls.Primitives.RangeBase.Value.

Свойства System.Windows.Controls.Primitives.RangeBase.SmallChange и System.Windows.Controls.Primitives.RangeBase.LargeChange могут интерпретироваться производными классами по-разному. Например, класс System.Windows.

Controls.ProgressBar их вообще не использует, а класс System.Windows.Controls.Primitives.ScrollBar использует одно значение при нажатии на стрелки полосы прокрутки, а другое при нажатии на саму полосу прокрутки.

## 7.1. Элемент управления ProgressBar

Элемент управления System.Windows.Controls.ProgressBar используется в ситуациях, требующих визуализации прогресса какой-либо операции.

Свойства класса System.Windows.Controls.ProgressBar:

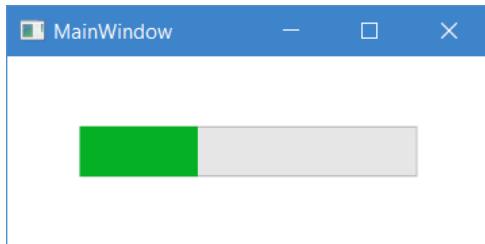
- **IsIndeterminate** (System.Boolean). Получает или задает значение, которое указывает, отображается ли прогресс с учетом текущего значения установленного в элементе или отображение прогресса происходит в общем виде. **True** если необходимо отображать прогресс в общем виде; иначе — **false**. Значение по умолчанию — **false**.
- **Orientation** (System.Windows.Controls.Orientation). Получает или задает ориентацию элемента (горизонтальную или вертикальную). Значение по умолчанию — System.Windows.Controls.Orientation.Horizontal.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке Wpf.Controls.ProgressBar.Xaml):

XAML

```
<ProgressBar Height="30" Maximum="100" Minimum="0"
             Value="35" Width="200"/>
```

Результат приведенной выше разметки показан на рис. 20.



**Рис. 20.** Элемент управления ProgressBar

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.ProgressBar.CSharp**):

C#

```
var progressBar = new ProgressBar
{
    Height = 30.0,
    Maximum = 100.0,
    Minimum = 0.0,
    Value = 35.0,
    Width = 200.0
};
```

## 7.2. Элемент управления Slider

Элемент управления **System.Windows.Controls.Slider** позволяет пользователю выбирать значение определенного параметра при помощи ползунка из допустимого диапазона значений.

Свойства класса **System.Windows.Controls.Slider**:

- **AutoToolTipPlacement** (**System.Windows.Controls.Primitives.AutoToolTipPlacement**). Получает или задает режим отображения подсказки при нажатии на ползунке, в которой отображается текущее выбран-

ное значение. При указании режима отображения также указывается место отображения. Значение по умолчанию — `System.Windows.Controls.Primitives.AutoToolTipPlacement.None`.

- **AutoToolTipPrecision** (`System.Int32`). Получает или задает количество цифр, которые необходимо отображать после десятичной точки, при отображении подсказки с текущим выбранным значением. Данное свойство может принимать только положительные значения. Значение по умолчанию — 0.
- **Delay** (`System.Int32`). Получает или задает количество миллисекунд, которое должно пройти перед тем как начать двигать ползунок, при нажатии на область слева или справа от ползунка. Значение по умолчанию — `System.Windows.SystemParameters.KeyboardDelay`.
- **Interval** (`System.Int32`). Получает или задает количество миллисекунд, которое должно пройти перед повторным смещением ползунка, при нажатии на область слева или справа от ползунка. Значение по умолчанию — `System.Windows.SystemParameters.KeyboardSpeed`.
- **IsDirectionReversed** (`System.Boolean`). Получает или задает значение, которое указывает направление, в котором увеличиваются значения. `True` если увеличение значений происходит влево, для горизонтально ориентированного ползунка, и вниз для вертикально ориентированного ползунка; иначе — `false`. Значение по умолчанию — `false`.
- **IsMoveToPointEnabled** (`System.Boolean`). Получает или задает значение, которое указывает, необходимо ли

перемещать ползунок в место щелчка мышью сразу. **True** если ползунок необходимо сразу перемещать в место щелчка мышью; иначе — **false**. Значение по умолчанию — **false**.

- **IsSelectionRangeEnabled** (System.Boolean). Получает или задает значение, которое указывает, необходимо ли отображать область выделения на элементе. **True** если область выделения необходимо отображать; иначе — **false**. Значение по умолчанию — **false**.
- **IsSnapToTickEnabled** (System.Boolean). Получает или задает значение, которое указывает необходимо ли перемещать ползунок между отметками или в свободном режиме. **True** если ползунок необходимо перемещать между отметками; иначе — **false**. Значение по умолчанию — **false**.
- **Orientation** (System.Windows.Controls.Orientation). Получает или задает ориентацию элемента (горизонтальную или вертикальную). Значение по умолчанию — System.Windows.Controls.Orientation.Horizontal.
- **SelectionEnd** (System.Double). Получает или задает максимальное значение области выделения на элементе. Значение по умолчанию — 0.0.
- **SelectionStart** (System.Double). Получает или задает минимальное значение области выделения на элементе. Значение по умолчанию — 0.0.
- **TickFrequency** (System.Double). Получает или задает частоту с которой располагаются отметки на элементе. Значение по умолчанию — 1.0.
- **TickPlacement** (System.Windows.Controls.Primitives.TickPlacement). Получает или задает режим отобра-

жения отметок на элементе. При указании режима отображения также указывается место отображения. Значение по умолчанию — System.Windows.Controls.Primitives.TickPlacement.None.

- **Ticks** (System.Windows.Media.DoubleCollection). Получает или задает коллекцию значений, для которых необходимо отображать отметки на элементе. Значение по умолчанию — пустая коллекция.

Для того, чтобы указать режим отображения подсказки при нажатии на ползунке необходимо использовать перечисление System.Windows.Controls.Primitives.AutoToolTipPlacement, которое содержит следующие варианты:

- BottomRight. Для горизонтально ориентированного ползунка подсказка отображается под ним. Для вертикально ориентированного ползунка подсказка отображается справа от него.
- None. Подсказка не показывается.
- TopLeft. Для горизонтально ориентированного ползунка подсказка отображается над ним. Для вертикально ориентированного ползунка подсказка отображается слева от него.

Для того, чтобы указать режим отображения отметок на ползунке необходимо использовать перечисление System.Windows.Controls.Primitives.TickPlacement, которое содержит следующие варианты:

- Both. Для горизонтально ориентированного ползунка отметки отображаются над и под ним. Для вертикально

ориентированного ползунка отметки отображаются слева и справа от него.

- BottomRight. Для горизонтально ориентированного ползунка отметки отображаются под ним. Для вертикально ориентированного ползунка отметки отображаются справа от него.
- None. Отметки не отображаются.
- TopLeft. Для горизонтально ориентированного ползунка отметки отображаются над ним. Для вертикально ориентированного ползунка отметки отображаются слева от него.

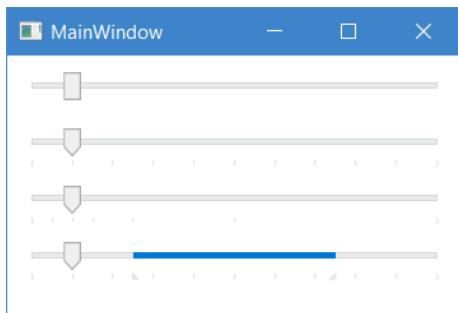
Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке **Wpf.Controls.Slider.Xaml**):

### XAML

```
<Grid Margin="10">
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Slider Grid.Row="0" Maximum="100" Value="10"/>
    <Slider Grid.Row="1"
            Maximum="100"
            TickFrequency="10"
            TickPlacement="BottomRight"
            Value="10"/>
    <Slider Grid.Row="2"
            Maximum="100"
            Ticks="0,5,10,15,25,50,100"
```

```
        TickPlacement="BottomRight"
        Value="10"/>
<Slider Grid.Row="3"
        IsSelectionRangeEnabled="True"
        Maximum="100"
        TickFrequency="10"
        TickPlacement="BottomRight"
        SelectionEnd="75"
        SelectionStart="25"
        Value="10"/>
</Grid>
```

Результат приведенной выше разметки показан на рис. 21.



**Рис. 21.** Элемент управления Slider

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.Slider.CSharp**):

**C#**

```
var slider1 = new Slider { Maximum = 100.0, Value = 10.0 };
var slider2 = new Slider
{
    Maximum = 100.0,
```

```
    TickFrequency = 10.0,
    TickPlacement = TickPlacement.BottomRight,
    Value = 10.0
};

var slider3 = new Slider
{
    Maximum = 100.0,
    TickPlacement = TickPlacement.BottomRight,
    Ticks = new DoubleCollection(new[] { 0.0, 5.0,
        10.0, 15.0, 25.0, 50.0, 100.0 }),
    Value = 10.0
};
var slider4 = new Slider
{
    IsSelectionRangeEnabled = true,
    Maximum = 100.0,
    SelectionEnd = 75.0,
    SelectionStart = 25.0,
    TickFrequency = 10.0,
    TickPlacement = TickPlacement.BottomRight,
    Value = 10.0
};
var grid = new Grid { Margin = new Thickness(10.0) };
grid.RowDefinitions.Add(new RowDefinition());
grid.RowDefinitions.Add(new RowDefinition());
grid.RowDefinitions.Add(new RowDefinition());
grid.RowDefinitions.Add(new RowDefinition());

Grid.SetRow(slider1, 0);
Grid.SetRow(slider2, 1);
Grid.SetRow(slider3, 2);
Grid.SetRow(slider4, 3);

grid.Children.Add(slider1);
grid.Children.Add(slider2);
grid.Children.Add(slider3);
grid.Children.Add(slider4);
```

## 7.3. Элемент управления ScrollBar

Элемент управления System.Windows.Controls.Primitives. **ScrollBar** представляет из себя полосу прокрутки, которую, обычно, используют для управления областью отображения, прокручиваемого содержимого.

Свойства класса System.Windows.Controls.Primitives. **ScrollBar**:

- **Orientation** (System.Windows.Controls. **Orientation**). Получает или задает ориентацию элемента (горизонтальную или вертикальную). Значение по умолчанию — System.Windows.Controls. **Orientation**.Vertical.
- **ViewportSize** (System. **Double**). Получает или задает размер видимого содержимого элемента, для которого используется текущая полоса прокрутки. Это значение используется для вычисления размера ползунка. Чем больше содержимого видно, тем больше размер ползунка. Значение по умолчанию — 0.0.

События класса System.Windows.Controls.Primitives. **ScrollBar**:

- **Scroll** (System.Windows.Controls.Primitives. **ScrollEventHandler**). Срабатывает, когда перемещается ползунок полосы прокрутки.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке **Wpf.Controls.ScrollBar.Xaml**):

### XAML

```
<Grid Margin="10">
    <Grid.ColumnDefinitions>
```

```
<ColumnDefinition/>
<ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
<RowDefinition/>
<RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<ScrollBar Grid.Column="1"
           Grid.Row="0"
           Maximum="50"
           Value="15"/>
<ScrollBar Grid.Column="0"
           Grid.Row="1"
           Maximum="30"
           Orientation="Horizontal"
           Value="10"/>
</Grid>
```

Результат приведенной выше разметки показан на рис. 22.

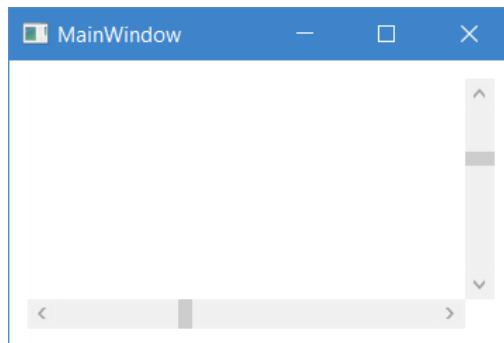


Рис. 22. Элемент управления ScrollBar

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.ScrollBar.CSharp**):

## C#

```
var scrollBar1 = new ScrollBar { Maximum = 50.0,
                               Value = 15.0 };
var scrollBar2 = new ScrollBar
{
    Maximum = 30.0,
    Orientation = Orientation.Horizontal,
    Value = 10.0
};

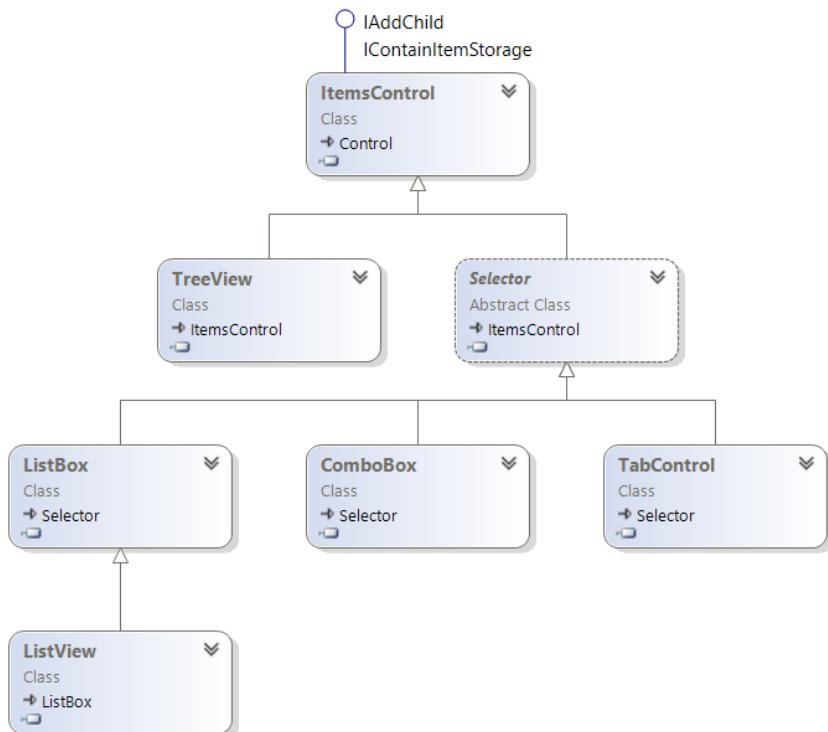
var grid = new Grid { Margin = new Thickness(10.0) };
grid.ColumnDefinitions.Add(new ColumnDefinition());
grid.ColumnDefinitions.Add(new ColumnDefinition {
    Width = GridLength.Auto });
grid.RowDefinitions.Add(new RowDefinition());
grid.RowDefinitions.Add(new RowDefinition {
    Height = GridLength.Auto });

Grid.SetColumn(scrollBar1, 1);
Grid.SetRow(scrollBar1, 0);
Grid.SetColumn(scrollBar2, 0);
Grid.SetRow(scrollBar2, 1);

grid.Children.Add(scrollBar1);
grid.Children.Add(scrollBar2);
```

# 8. Списковые элементы управления

В WPF существует группа элементов управления предназначенных для работы с набором (список) объектом. Базовым классом для всех них является класс `System.Windows.Controls.ItemsControl` (рис. 23), который предоставляет базовую функциональность для хранения



**Рис. 23.** Диаграмма классов, описывающих списковые элементы управления

и обработки коллекции объектов. Те элементы управления, которые хранят объекты, помещенные в них в виде одноуровневой коллекции, наследуются от другого промежуточного класса — `System.Windows.Controls.Primitives.Selector`, который предоставляет функциональность для обработки выделений объектов.

Свойства класса `System.Windows.Controls.Primitives.Selector`:

- **SelectedIndex** (`System.Int32`). Получает или задает индекс первого объекта в текущем выделении. Если выделение отсутствует, используется значение `-1`. Значение по умолчанию — `-1`.
- **SelectedItem** (`System.Object`). Получает или задает первый объект в текущем выделении. Если выделение отсутствует, используется значение `null`. Значение по умолчанию — `null`.

События класса `System.Windows.Controls.Primitives.Selector`:

- **SelectionChanged** (`System.Windows.Controls.SelectionChangedEventHandler`). Срабатывает, когда состояние выделения в элементе изменяется.

Модель управления содержимым предоставляемая классом `System.Windows.Controls.ItemsControl` работает таким образом, что каждый элемент, помещенный в коллекцию дочерних элементов, «обворачивается» в специальный контейнер элемента. У каждого спискового элемента управления существует собственный вид данного контейнера. При необходимости более тонкой настройки контейнера элемента, есть возможность описать его создание

явным образом. На рис. 24 показаны классы контейнеров элементов, предназначенных специально для того, чтобы использоваться в качестве «оберточ» над содержимым каждого элемента списка.

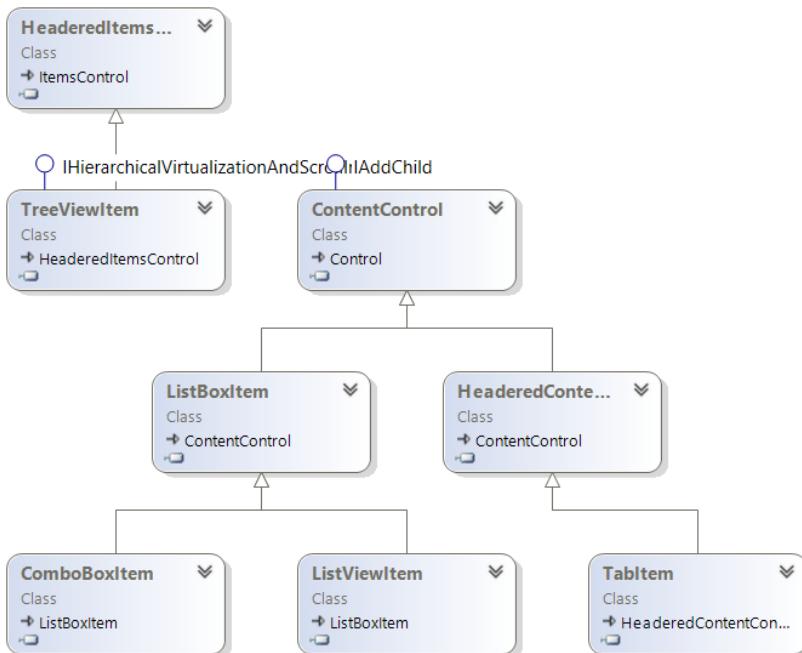


Рис. 24. Диаграмма классов, описывающих контейнеры элементов

## 8.1. Элемент управления **ListBox**

Элемент управления `System.Windows.Controls.ListBox` предназначен для отображения списка объектов. Если объектов, помещенных в элемент, становится слишком много, появляются полосы прокрутки для навигации по содержимому. Также элемент поддерживает несколько режимов выделения содержимого.

Свойства класса System.Windows.Controls.ListBox:

- **SelectedItems** (System.Collections.IList). Получает коллекцию объектов текущего выделения. Значение по умолчанию — пустая коллекция.
- **SelectionMode** (System.Windows.Controls.SelectionMode). Получает или задает режим выделения объектов в элементе. Значение по умолчанию — System.Windows.Controls.SelectionMode.Single.

Методы класса System.Windows.Controls.ListBox:

- **ScrollIntoView(item)**. Прокручивает содержимое элемента таким образом, чтобы указанный объект оказался в области отображения.
- **SelectAll()**. Выделяет все объекты в элементе.
- **UnselectAll()**. Снимает выделение со всех объектов в элементе.

Для того, чтобы указать режим выделения объектов необходимо использовать перечисление System.Windows.Controls.SelectionMode, которое содержит следующие варианты:

- **Extended**. Можно выделять более одного объекта. Нажав клавишу *Shift* можно выделить более одного элемента за раз.
- **Multiple**. Можно выделять более одного объекта. Выделение с нажатой клавишей *Shift* не поддерживается.
- **Single**. Можно выделять не более одного объекта за раз.

В качестве контейнера элементов для спискового элемента управления System.Windows.Controls.ListBox применяется класс System.Windows.Controls.ListBoxItem.

Свойства класса System.Windows.Controls.ListBoxItem:

- **IsSelected** (System.Boolean). Получает или задает значение, которое указывает, выделен ли элемент. **True** если элемент выделен; иначе — **false**. Значение по умолчанию — **false**.

События класса System.Windows.Controls.ListBoxItem:

- **Selected** (System.Windows.RoutedEventHandler). Срабатывает, когда элемент выделяется.
- **Unselected** (System.Windows.RoutedEventHandler). Срабатывает, когда элемент теряет выделение.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке **Wpf.Controls.ListBox.Xaml**):

### XAML

```
<ListBox Height="100" SelectionMode="Multiple"
         Width="200">
    <ListBoxItem>Red</ListBoxItem>
    <ListBoxItem IsSelected="True">Green</ListBoxItem>
    <ListBoxItem IsSelected="True">Blue</ListBoxItem>
</ListBox>
```

Результат приведенной выше разметки показан на рис. 25.

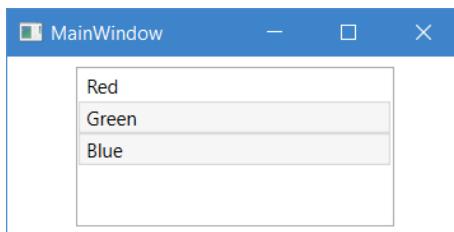


Рис. 25. Элемент управления ListBox

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.ListBox.CSharp**):

C#

```
var item1 = new ListBoxItem { Content = "Red" };
var item2 = new ListBoxItem { Content = "Green",
    IsSelected = true };
var item3 = new ListBoxItem { Content = "Blue",
    IsSelected = true };
var listBox = new ListBox
{
    Height = 100.0,
    SelectionMode = SelectionMode.Multiple,
    Width = 200.0
};
listBox.Items.Add(item1);
listBox.Items.Add(item2);
listBox.Items.Add(item3);
```

## 8.2. Элемент управления ListView

Элемент управления `System.Windows.Controls.ListView` предназначен для отображения списка объектов. Этот элемент, в отличии от элемента управления `System.Windows.Controls.ListBox`, способен отображать помещенные в него объекты в виде таблицы с именованными колонками. Для реализации табличного вида необходимо использовать привязку данных, которую рассмотрим в будущих разделах.

## 8.3. Элемент управления ComboBox

Элемент управления `System.Windows.Controls.ComboBox` отличается от других списковых элементов управления тем, что он отображает не более одного выделенного

объекта. Остальные объекты скрываются в выпадающем меню, которое отображается при нажатии на элемент.

Свойства класса System.Windows.Controls.**ComboBox**:

- **IsDropDownOpen** (System.Boolean). Получает или задает значение, которое указывает, открыто ли выпадающее меню. **True** если выпадающее меню открыто; иначе — **false**. Значение по умолчанию — **false**.
- **IsEditable** (System.Boolean). Получает или задает значение, которое указывает, выбранное значение будет отображаться в виде кнопки или поля для ввода текста. **True** если элемент отображает поле для ввода текста; иначе — **false**. Значение по умолчанию — **false**.
- **IsReadOnly** (System.Boolean). Получает или задает значение, которое указывает можно ли редактировать текст в поле для ввода текста (если, конечно же, включен режим его отображения). **True** текст нельзя редактировать; иначе — **false**. Значение по умолчанию — **false**.
- **MaxDropDownHeight** (System.Double). Получает или задает максимальную высоту выпадающего меню.
- **ShouldPreserveUserEnteredPrefix** (System.Boolean). Получает или задает значение, которое указывает, необходимо ли оставлять введенный пользователем текст в оригинальном виде или стоит его заменить на значение существующего объекта. **True** если необходимо оставить введенное пользователем значение; иначе — **false**. Значение по умолчанию — **false**.
- **StaysOpenOnEdit** (System.Boolean). Получает или задает значение, которое указывает, необходимо ли оставлять открытым выпадающий список после того, как пользователь щелкнет на поле для ввода текста. **True**

если выпадающий список должен оставаться открытым; иначе — **false**. Значение по умолчанию — **false**.

- **Text** (System.String). Получает или задает текст текущего выделенного объекта. Значение по умолчанию — System.String.Empty.

События класса System.Windows.Controls.ComboBox:

- **DropDownClosed** (System.EventHandler). Срабатывает, когда выпадающий список закрывается.
- **DropDownOpened** (System.EventHandler). Срабатывает, когда выпадающий список открывается.

В качестве контейнера элементов для спискового элемента управления System.Windows.Controls.ComboBox применяется класс System.Windows.Controls.ComboBoxItem.

Свойства класса System.Windows.Controls.ComboBoxItem:

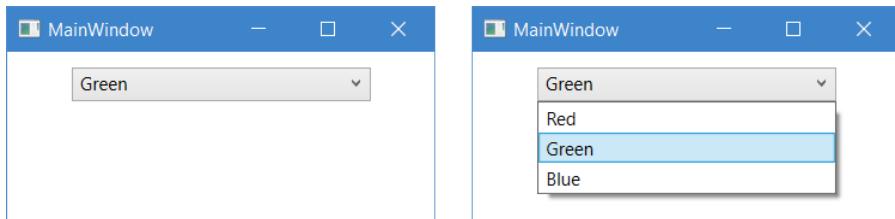
- **IsHighlighted** (System.Boolean). Получает или задает значение, которое указывает, подсвечен ли элемент. **True** если элемент подсвечен; иначе — **false**. Значение по умолчанию — **false**.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке Wpf.Controls.ComboBox.Xaml):

### XAML

```
<ComboBox Height="23" Width="200">
    <ComboBoxItem>Red</ComboBoxItem>
    <ComboBoxItem IsSelected="True">Green</ComboBoxItem>
    <ComboBoxItem>Blue</ComboBoxItem>
</ComboBox>
```

Результат приведенной выше разметки показан на рис. 26.



**Рис. 26.** Элемент управления ComboBox

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.ComboBox.CSharp**):

C#

```
var item1 = new ComboBoxItem { Content = "Red" };
var item2 = new ComboBoxItem { Content = "Green",
    IsSelected = true };
var item3 = new ComboBoxItem { Content = "Blue" };

var comboBox = new ComboBox { Height = 23.0,
    Width = 200.0 };
comboBox.Items.Add(item1);
comboBox.Items.Add(item2);
comboBox.Items.Add(item3);
```

## 8.4. Элемент управления TreeView

Элемент управления `System.Windows.Controls.TreeView` предназначен для отображения иерархической (деревовидной) структуры данных. Каждый объект, помещенный в элемент, является узлом, который обладает собственным значением и коллекцией дочерних узлов.

Свойства класса System.Windows.Controls.[TreeView](#):

- **SelectedItem** (System.Object). Получает или задает выделенный объект в элементе. Если выделение отсутствует, используется значение `null`. Значение по умолчанию — `null`.

События класса System.Windows.Controls.[TreeView](#):

- **SelectedItemChanged** (System.Windows.RoutedPropertyChangedEventHandler<System.Object>). Срабатывает, когда состояние выделения в элементе изменяется.

Описание элементов, помещенных в списковый элемент управления System.Windows.Controls.[TreeView](#), отличается от рассмотренных ранее. Отличие заключается в том, что элементы System.Windows.Controls.[ListBox](#) и System.Windows.Controls.[ComboBox](#) содержат одноуровневую коллекцию элементов, предназначенных для отображения. Элемент System.Windows.Controls.[TreeView](#) описывает иерархическую структуру элементов. Это означает, что каждый элемент, помимо собственно своего содержимого, может также содержать набор других (дочерних) элементов, устроенных по такому же принципу. Для описания такой структуры используется класс System.Windows.Controls.[TreeViewItem](#).

Свойства класса System.Windows.Controls.[TreeViewItem](#):

- **IsExpanded** (System.Boolean). Получает или задает значение, которое указывает, раскрыты ли дочерние элементы. `True` если дочерние элементы раскрыты; иначе — `false`. Значение по умолчанию — `false`.
- **isSelected** (System.Boolean). Получает или задает значение, которое указывает, выделен ли элемент. `True`

если элемент выделен; иначе — **false**. Значение по умолчанию — **false**.

- **IsSelectionActive** (System.Boolean). Получает значение, которое указывает, содержит ли элемент фокус ввода. **True** если элемент содержит фокус ввода; иначе — **false**. Значение по умолчанию — **false**.

События класса System.Windows.Controls.TreeViewItem:

- **Collapsed** (System.Windows.RoutedEventArgs). Срабатывает, когда элемент скрывает свои дочерние элементы.
- **Expanded** (System.Windows.RoutedEventArgs). Срабатывает, когда элемент раскрывает свои дочерние элементы.
- **Selected** (System.Windows.RoutedEventArgs). Срабатывает, когда элемент выделяется.
- **Unselected** (System.Windows.RoutedEventArgs). Срабатывает, когда элемент теряет выделение.

Методы класса System.Windows.Controls.TreeViewItem:

- **ExpandSubtree()**. Раскрывает текущий элемент и все его дочерние элементы.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке **Wpf.Controls.TreeView.Xaml**):

### XAML

```
<TreeView>
    <TreeViewItem Header="Europe">
        <TreeViewItem Header="England">
            <TreeViewItem Header="London"/>
            <TreeViewItem Header="Liverpool"/>
        </TreeViewItem>
```

```
<TreeViewItem Header="France">
    <TreeViewItem Header="Marseille"/>
</TreeViewItem>
</TreeViewItem>
<TreeViewItem Header="North America">
    <TreeViewItem Header="Canada">
        <TreeViewItem Header="Toronto"/>
        <TreeViewItem Header="Montreal"
                      IsSelected="True"/>
        <TreeViewItem Header="Kingston"/>
    </TreeViewItem>
</TreeViewItem>
<TreeViewItem Header="South America">
    <TreeViewItem Header="Argentina">
        <TreeViewItem Header="Buenos Aires"/>
        <TreeViewItem Header="Rosario"/>
    </TreeViewItem>
</TreeViewItem>
</TreeView>
```

Результат приведенной выше разметки показан на рис. 27.

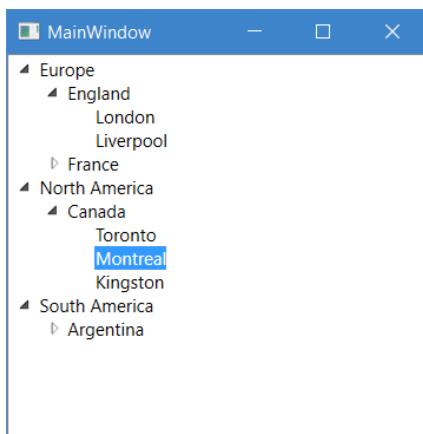


Рис. 27. Элемент управления TreeView

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.TreeView.CSharp**):

C#

```

var item31 = new TreeViewItem { Header = "London" };
var item32 = new TreeViewItem { Header = "Liverpool" };
var item33 = new TreeViewItem { Header = "Marseille" };
var item34 = new TreeViewItem { Header = "Toronto" };
var item35 = new TreeViewItem { Header = "Montreal",
    IsSelected = true };
var item36 = new TreeViewItem { Header = "Kingston" };
var item37 = new TreeViewItem { Header = "Buenos
    Aires" };
var item38 = new TreeViewItem { Header = "Rosario" };
var item21 = new TreeViewItem { Header = "England" };
item21.Items.Add(item31);
item21.Items.Add(item32);
var item22 = new TreeViewItem { Header = "France" };
item22.Items.Add(item33);
var item23 = new TreeViewItem { Header = "Canada" };
item23.Items.Add(item34);
item23.Items.Add(item35);
item23.Items.Add(item36);
var item24 = new TreeViewItem { Header = "Argentina" };
item24.Items.Add(item37);
item24.Items.Add(item38);
var item11 = new TreeViewItem { Header = "Europe" };
item11.Items.Add(item21);
item11.Items.Add(item22);
var item12 = new TreeViewItem { Header = "North
    America" };
item12.Items.Add(item23);
var item13 = new TreeViewItem { Header = "South
    America" };
item13.Items.Add(item24);
var treeView = new TreeView();

```

```
treeView.Items.Add(item11);
treeView.Items.Add(item12);
treeView.Items.Add(item13);
```

## 8.5. Элемент управления TabControl

Элемент управления System.Windows.Controls.**TabControl** предназначен для отображения содержимого в виде закладок. Поначалу может показаться, что у данного элемента нет ничего общего с предыдущими списковыми элементами управления, но это не так. Данный элемент управления может отображать только одну из выбранных закладок, которые, по сути, являются списком доступных для отображения элементов. Именно поэтому базовым классом для него является класс System.Windows.Controls.Primitives.**Selector**, который поддерживает функциональность выделения одного элемента из набора доступных.

Свойства класса System.Windows.Controls.**TabControl**:

- **TabStripPlacement** (System.Windows.Controls.**Dock**). Получает или задает режим выравнивания полоски с заголовками закладок относительно содержимого активной закладки. Значение по умолчанию — System.Windows.Controls.**Dock**.Top.

В качестве контейнера элементов для спискового элемента управления System.Windows.Controls.**TabControl** применяется класс System.Windows.Controls.**TabItem**.

Свойства класса System.Windows.Controls.**TabItem**:

- **IsSelected** (System.Boolean). Получает или задает значение, которое указывает, выделен ли текущий элемент.

**True** если текущий элемент выделен; иначе — **false**.  
Значение по умолчанию — **false**.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке **Wpf.Controls.TabControl.Xaml**):

### XAML

```
<TabControl>
    <TabItem Header="Colors">
        <ListBox Margin="10">
            <ListBoxItem>Red</ListBoxItem>
            <ListBoxItem>Green</ListBoxItem>
            <ListBoxItem>Blue</ListBoxItem>
        </ListBox>
    </TabItem>
    <TabItem Header="Cities" IsSelected="True">
        <ListBox Margin="10">
            <ListBoxItem>London</ListBoxItem>
            <ListBoxItem>Paris</ListBoxItem>
            <ListBoxItem>Berlin</ListBoxItem>
        </ListBox>
    </TabItem>
</TabControl>
```

Результат приведенной выше разметки показан на рис. 28.

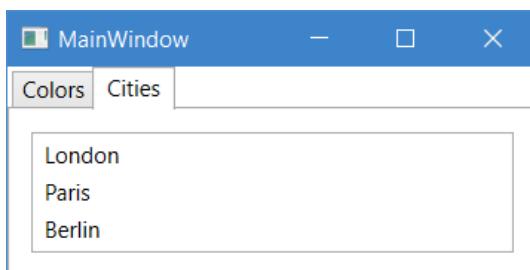


Рис. 28. Элемент управления TabControl

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.TabControl.CSharp**):

C#

```
var item1 = new ListBoxItem { Content = "Red" };
var item2 = new ListBoxItem { Content = "Green" };
var item3 = new ListBoxItem { Content = "Blue" };

var listBox1 = new ListBox { Margin =
    new Thickness(10.0) };
listBox1.Items.Add(item1);
listBox1.Items.Add(item2);
listBox1.Items.Add(item3);

var item4 = new ListBoxItem { Content = "London" };
var item5 = new ListBoxItem { Content = "Paris" };
var item6 = new ListBoxItem { Content = "Berlin" };

var listBox2 = new ListBox { Margin =
    new Thickness(10.0) };
listBox2.Items.Add(item4);
listBox2.Items.Add(item5);
listBox2.Items.Add(item6);

var tabItem1 = new TabItem { Content = listBox1,
    Header = "Colors" };
var tabItem2 = new TabItem { Content = listBox2,
    Header = "Cities", IsSelected = true };

var tabControl = new TabControl();
tabControl.Items.Add(tabItem1);
tabControl.Items.Add(tabItem2);
```

# 9. Всплывающие окна

Элемент управления System.Windows.Controls.Primitives. **Popup** представляет из себя всплывающее окно с содержимым. Его можно использовать для того, чтобы отобразить часть пользовательского интерфейса в отдельном окне, которое показывается поверх текущего окна приложения относительно назначенного элемента или точки на экране.

Свойства класса System.Windows.Controls.Primitives. **Popup**:

- **CustomPopupPlacementCallback** (System.Windows.Controls.Primitives.[CustomPopupPlacementCallback](#)). Получает или задает метод обратного вызова, который используется для определения позиционирования всплывающего окна, если значение свойства System.Windows.Controls.[ContextMenu](#).Placement **равно** System.Windows.Controls.Primitives.[PlacementMode](#).Custom.
- **HasDropShadow** (System.Boolean). Получает или задает значение, которое указывает, должно ли всплывающее окно отбрасывать тень. [True](#) если всплывающее окно должно отбрасывать тень; иначе — [false](#). Значение по умолчанию — [false](#).
- **HorizontalOffset** (System.Double). Получает или задает горизонтальный отступ относительно целевой области. Значение по умолчанию — 0.0.
- **IsOpen** (System.Boolean). Получает или задает значение, которое указывает, отображается ли всплывающее

окно. **True** если всплывающее окно отображается; иначе — **false**. Значение по умолчанию — **false**.

- **Placement** ([System.Windows.Controls.Primitives.PlacementMode](#)). Получает или задает режим расположения всплывающего окна. Значение по умолчанию — [System.Windows.Controls.Primitives.PlacementMode.MousePoint](#).
- **PlacementRectangle** ([System.Windows.Rect](#)). Получает или задает область, относительно которой позиционируется всплывающее окно при отображении. Значение по умолчанию — [System.Windows.Rect.Empty](#).
- **PlacementTarget** ([System.Windows.UIElement](#)). Получает или задает элемент, относительно которого позиционируется всплывающее окно при отображении. Значение по умолчанию — **null**.
- **StaysOpen** ([System.Boolean](#)). Получает или задает значение, которое указывает, должно ли всплывающее окно закрываться автоматически. **True** если всплывающее окно должно оставаться открытым до тех пор, пока данному свойству не будет присвоено значение **false**; иначе — **false**. Значение по умолчанию — **false**.
- **VerticalOffset** ([System.Double](#)). Получает или задает вертикальный отступ относительно целевой области. Значение по умолчанию — 0.0.

События класса [System.Windows.Controls.Primitives.Popup](#):

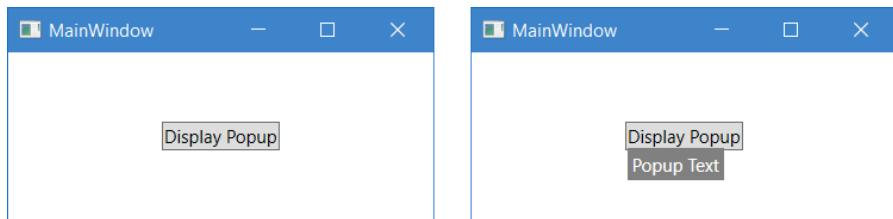
- **Closed** ([System.Windows.RoutedEventArgs](#)). Срабатывает, когда всплывающее окно закрывается.
- **Opened** ([System.Windows.RoutedEventArgs](#)). Срабатывает, когда всплывающее окно открывается.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке **Wpf.Controls.Popup.Overview.Xaml**):

### XAML

```
<Button Click="Button_Click">
    <StackPanel>
        <TextBlock Text="Display Popup"/>
        <Popup x:Name="popup">
            <TextBlock Background="Gray"
                       Foreground="White"
                       Padding="3"
                       Text="Popup Text"/>
        </Popup>
    </StackPanel>
</Button>
```

Результат приведенной выше разметки показан на рис. 29.



**Рис. 29.** Элемент управления Popup

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.Popup.Overview.CSharp**):

### C#

```
var textBlock1 = new TextBlock
{
```

```

        Background = Brushes.Gray,
        Foreground = Brushes.White,
        Padding = new Thickness(3.0),
        Text = "Popup Text"
    };
var popup = new Popup { Child = textBlock1 };
var textBlock2 = new TextBlock { Text =
    "Display Popup" };
var stackPanel = new StackPanel();
stackPanel.Children.Add(textBlock2);
stackPanel.Children.Add(popup);
var button = new Button { Content = stackPanel };
button.Click += (sender, e) => popup.IsOpen = true;

```

Элемент управления `System.Windows.Controls.Primitives.Popup` содержит базовую функциональность для описания всплывающих окон, и его стоит использовать при необходимости создания нового элемента управления или описания шаблона для существующего. Для наиболее распространенных видов всплывающих окон в WPF присутствует ряд готовых классов:

- `System.Windows.Controls.ToolTip`. Подсказка.
- `System.Windows.Controls.ContextMenu`. Контекстное меню.
- `System.Windows.Controls.ComboBox`. Выпадающий список.

## 9.1. Позиционирование всплывающих окон

Всплывающее окно может быть позиционировано относительно элемента управления, курсора мыши или экрана, используя следующий набор свойств: `System.Windows.Controls.Primitives.Popup.Placement`, `System.`

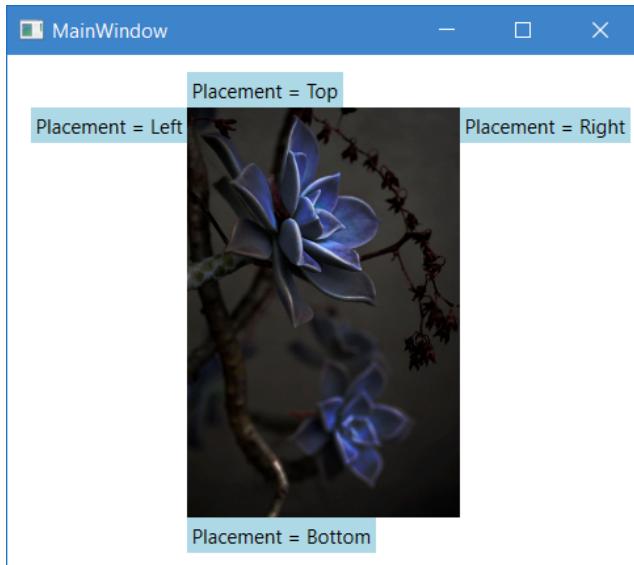
Windows.Controls.Primitives.Popup.PlacementTarget, System.Windows.Controls.Primitives.Popup.PlacementRectangle, System.Windows.Controls.Primitives.Popup.HorizontalOffset и System.Windows.Controls.Primitives.Popup.VerticalOffset.

Приведенный ниже фрагмент разметки демонстрирует как можно позиционировать всплывающие окна, пользуясь только свойством System.Windows.Controls.Primitives.Popup.Placement (полный пример находится в папке **Wpf.Controls.Popup.Placement.Xaml**):

### XAML

```
<Grid HorizontalAlignment="Center"
      VerticalAlignment="Center">
    <Image Height="250" Source="Succulent.jpg"/>
    <Popup IsOpen="True" Placement="Bottom">
        <TextBlock Background="LightBlue" Padding="3"
                   Text="Placement = Bottom"/>
    </Popup>
    <Popup IsOpen="True" Placement="Left">
        <TextBlock Background="LightBlue" Padding="3"
                   Text="Placement = Left"/>
    </Popup>
    <Popup IsOpen="True" Placement="Right">
        <TextBlock Background="LightBlue" Padding="3"
                   Text="Placement = Right"/>
    </Popup>
    <Popup IsOpen="True" Placement="Top">
        <TextBlock Background="LightBlue" Padding="3"
                   Text="Placement = Top"/>
    </Popup>
</Grid>
```

Результат приведенной выше разметки показан на рис. 30.



**Рис. 30.** Элемент управления PopUp, позиционированный относительно родительского элемента

В таком случае всплывающее окно будет позиционироваться относительно своего родительского элемента.

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.Popup.Placement.CSharp**):

C#

```
var image = new Image
{
    Height = 250.0,
    Source = new BitmapImage(new Uri("/Succulent.
        jpg", UriKind.Relative))
};

var textBlock1 = new TextBlock
{
```

```
Background = Brushes.LightBlue,
Padding = new Thickness(3.0),
Text = "Placement = Bottom"
};

var popup1 = new Popup { Child = textBlock1,
                      Placement = PlacementMode.Bottom };
var textBlock2 = new TextBlock
{
    Background = Brushes.LightBlue,
    Padding = new Thickness(3.0),
    Text = "Placement = Left"
};
var popup2 = new Popup { Child = textBlock2,
                      Placement = PlacementMode.Left };
var textBlock3 = new TextBlock
{
    Background = Brushes.LightBlue,
    Padding = new Thickness(3.0),
    Text = "Placement = Right"
};
var popup3 = new Popup { Child = textBlock3,
                      Placement = PlacementMode.Right };
var textBlock4 = new TextBlock
{
    Background = Brushes.LightBlue,
    Padding = new Thickness(3.0),
    Text = "Placement = Top"
};
var popup4 = new Popup { Child = textBlock4,
                      Placement = PlacementMode.Top };

var grid = new Grid
{
    HorizontalAlignment = HorizontalAlignment.Center,
    VerticalAlignment = VerticalAlignment.Center
};
```

```
grid.Children.Add(image);  
grid.Children.Add(popup1);  
grid.Children.Add(popup2);  
grid.Children.Add(popup3);  
grid.Children.Add(popup4);
```

Чтобы в дальнейшем объяснение свойств позиционирования всплывающих окон было понятнее, необходимо ввести несколько понятий: целевой объект (*target object*), целевая область (*target area*), целевая точка начала (*target origin*) и точка выравнивания всплывающего окна (*popup alignment point*). Данная терминология предоставит удобный способ обращения к различным аспектам всплывающего окна и ассоциированному с ним элементу управления.

### 9.1.1. Целевой объект

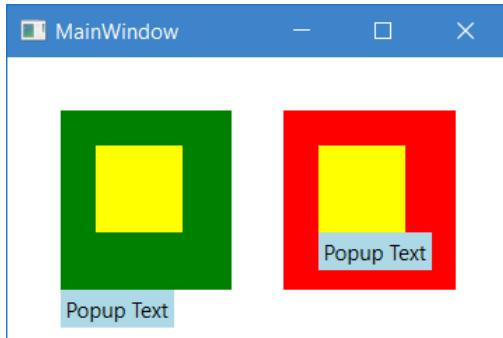
Целевым объектом называется элемент управления, ассоциированный со всплывающим окном. Если установлено значение свойству System.Windows.Controls.Primitives.Popup.PlacementTarget, то оно описывает целевой объект. Если данное свойство не установлено, и всплывающее окно имеет родительский элемент, то родительский элемент является целевым объектом. Если не задано значение свойству System.Windows.Controls.Primitives.Popup.PlacementTarget и всплывающее окно не обладает родительским элементом, то оно позиционируется относительно экрана.

Приведенный ниже фрагмент разметки демонстрирует создание всплывающего окна с явным указанием целевого объекта и без него (полный пример находится в папке Wpf.Controls.Popup.TargetObject.Xaml):

## XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Canvas Background="Green"
        Grid.Column="0"
        Margin="30,30,15,30">
        <Border Background="Yellow"
            Canvas.Left="20"
            Canvas.Top="20"
            Height="50"
            Width="50"/>
        <Popup IsOpen="True">
            <TextBlock Background="LightBlue"
                Padding="3"
                Text="Popup Text"/>
        </Popup>
    </Canvas>
    <Canvas Background="Red"
        Grid.Column="1"
        Margin="15,30,30,30">
        <Border x:Name="border"
            Background="Yellow"
            Canvas.Left="20"
            Canvas.Top="20"
            Height="50"
            Width="50"/>
        <Popup IsOpen="True"
            PlacementTarget="{Binding
                ElementName=border}">
            <TextBlock Background="LightBlue"
                Padding="3"
                Text="Popup Text"/>
        </Popup>
    </Canvas>
</Grid>
```

Результат приведенной выше разметки показан на рис. 31.



**Рис. 31.** Элемент управления Popup с указанием целевого объекта и без указания

Для обоих всплывающих окон не было установлено свойство `System.Windows.Controls.Primitives.Popup.Placement`, поэтому было взято его значение по умолчанию, которое указывает, что всплывающее окно должно позиционироваться под целевым объектом. Для первого всплывающего окна свойство `System.Windows.Controls.Primitives.Popup.PlacementTarget` не было установлено, поэтому в качестве целевого объекта был выбран родительский элемент `<Canvas>`. Для второго всплывающего окна в качестве целевого объекта был установлен элемент `<Border>`.

Рассмотренной выше разметке соответствует следующий код (полный пример находится в `Wpf.Controls.Popup.TargetObject.CSharp`):

C#

```
var border1 = new Border { Background = Brushes.Yellow,
                           Height = 50.0, Width = 50.0 };
var textBlock1 = new TextBlock
```

```
{  
    Background = Brushes.LightBlue,  
    Padding = new Thickness(3.0),  
    Text = "Popup Text"  
};  
  
var popup1 = new Popup { Child = textBlock1 };  
var canvas1 = new Canvas  
{  
    Background = Brushes.Green,  
    Margin = new Thickness(30.0, 30.0, 15.0, 30.0)  
};  
  
canvas1.Children.Add(border1);  
canvas1.Children.Add(popup1);  
Canvas.SetLeft(border1, 20.0);  
Canvas.SetTop(border1, 20.0);  
popup1.isOpen = true;  
  
var border2 = new Border { Background = Brushes.  
    Yellow, Height = 50.0, Width = 50.0 };  
  
var textBlock2 = new TextBlock  
{  
    Background = Brushes.LightBlue,  
    Padding = new Thickness(3.0),  
    Text = "Popup Text"  
};  
  
var popup2 = new Popup { Child = textBlock2,  
    PlacementTarget = border2 };  
  
var canvas2 = new Canvas  
{  
    Background = Brushes.Red,  
    Margin = new Thickness(15.0, 30.0, 30.0, 30.0)  
};
```

```
canvas2.Children.Add(border2);
canvas2.Children.Add(popup2);

Canvas.SetLeft(border2, 20.0);
Canvas.SetTop(border2, 20.0);

popup2.IsOpen = true;

var grid = new Grid();
grid.ColumnDefinitions.Add(new ColumnDefinition());
grid.ColumnDefinitions.Add(new ColumnDefinition());

grid.Children.Add(canvas1);
grid.Children.Add(canvas2);

Grid.SetColumn(canvas1, 0);
Grid.SetColumn(canvas2, 1);
```

### 9.1.2. Целевая область

Целевой областью является область на экране, относительно которой происходит позиционирование всплывающего окна. В рассмотренном ранее примере целевой областью являлась область, ограничивающая целевой объект. Однако, могут быть ситуации, когда установлен целевой объект, но при этом целевая область отличается от области, занимаемой им. Для указания целевой области необходимо использовать свойство `System.Windows.Controls.Primitives.Popup.PlacementRectangle`.

Приведенный ниже фрагмент разметки демонстрирует создание всплывающего окна с явным указанием целевой области и без нее (полный пример находится в папке `Wpf.Controls.Popup.TargetArea.Xaml`):

**XAML**

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <Canvas Background="Green"
        Grid.Column="0"
        Margin="30,30,15,30">
        <Border Background="Yellow"
            Canvas.Left="20"
            Canvas.Top="20"
            Height="50"
            Width="50"/>
        <Popup IsOpen="True">
            <TextBlock Background="LightBlue"
                Padding="3" Text="Popup Text"/>
        </Popup>
    </Canvas>

    <Canvas Background="Red"
        Grid.Column="1"
        Margin="15,30,30,30">
        <Border Background="Yellow"
            Canvas.Left="20"
            Canvas.Top="20"
            Height="50"
            Width="50"/>
        <Popup IsOpen="True"
            PlacementRectangle="20,20,60,60">
            <TextBlock Background="LightBlue"
                Padding="3"
                Text="Popup Text"/>
        </Popup>
    </Canvas>
</Grid>
```

Результат приведенной выше разметки показан на рис. 32.

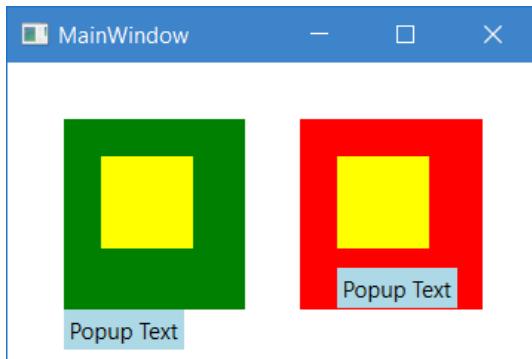


Рис. 32. Элемент управления Popup  
с указанием целевой области и без указания

В данном примере создается два элемента `<Canvas>`, каждый из которых содержит элемент рамку (`<Border>`) и всплывающее окно (`<Popup>`). В обоих случаях целевым объектом для всплывающего окна является панель, в которую оно помещено. В первом случае свойство `System.Windows.Controls.Primitives.Popup.PlacementRectangle` не задано, поэтому целевой областью является все пространство, занимаемое целевым объектом, т.е. панелью. Во втором случае целевая область задана явным образом. Стоит заметить, что координаты, описывающие целевую область, задаются относительно целевого объекта, а именно — его верхнего левого угла. Также стоит заметить, что само по себе задание целевой области никаким образом не визуализируется.

Рассмотренной выше разметке соответствует следующий код (полный пример находится в `Wpf.Controls.Popup.TargetArea.CSharp`):

## C#

```
var border1 = new Border { Background = Brushes.  
    Yellow, Height = 50.0, Width = 50.0 };  
var textBlock1 = new TextBlock  
{  
    Background = Brushes.LightBlue,  
    Padding = new Thickness(3.0),  
    Text = "Popup Text"  
};  
var popup1 = new Popup { Child = textBlock1 };  
var canvas1 = new Canvas  
{  
    Background = Brushes.Green,  
    Margin = new Thickness(30.0, 30.0, 15.0, 30.0)  
};  
canvas1.Children.Add(border1);  
canvas1.Children.Add(popup1);  
Canvas.SetLeft(border1, 20.0);  
Canvas.SetTop(border1, 20.0);  
popup1.IsOpen = true;  
var border2 = new Border { Background = Brushes.  
    Yellow, Height = 50.0, Width = 50.0 };  
var textBlock2 = new TextBlock  
{  
    Background = Brushes.LightBlue,  
    Padding = new Thickness(3.0),  
    Text = "Popup Text"  
};  
var popup2 = new Popup  
{  
    Child = textBlock2,  
    PlacementRectangle = new Rect(20.0, 20.0, 60.0, 60.0)  
};  
var canvas2 = new Canvas  
{  
    Background = Brushes.Red,  
    Margin = new Thickness(15.0, 30.0, 30.0, 30.0)  
};
```

```
canvas2.Children.Add(border2);
canvas2.Children.Add(popup2);
Canvas.SetLeft(border2, 20.0);
Canvas.SetTop(border2, 20.0);

popup2.IsOpen = true;

var grid = new Grid();
grid.ColumnDefinitions.Add(new ColumnDefinition());
grid.ColumnDefinitions.Add(new ColumnDefinition());

grid.Children.Add(canvas1);
grid.Children.Add(canvas2);

Grid.SetColumn(canvas1, 0);
Grid.SetColumn(canvas2, 1);
```

### **9.1.3. Целевая точка начала и точка выравнивания всплывающего окна**

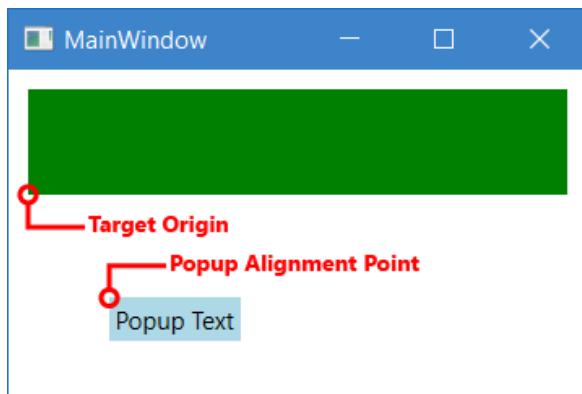
Целевая точка начала и точка выравнивания всплывающего окна — это точки на целевой области и всплывающем окне соответственно, которые используются для позиционирования всплывающего окна относительно целевой области. Для того, чтобы указать смещение одной точки относительно другой необходимо использовать свойства System.Windows.Controls.Primitives.Popup.HorizontalOffset и System.Windows.Controls.Primitives.Popup.VerticalOffset, которые задают горизонтальное и вертикальное смещение соответственно. Эти величины описывают насколько смещена точка выравнивания всплывающего окна относительно целевой точки начала. Точное местоположение данных точек определяется свойством System.Windows.Controls.Primitives.Popup.Placement.

Приведенный ниже фрагмент разметки демонстрирует создание всплывающего окна с указанием смещения его точки выравнивания (полный пример находится в папке Wpf.Controls.Popup.Offsets.Xaml):

### XAML

```
<Grid>
    <Canvas Background="Green" Margin="10,10,10,100">
        <Popup IsOpen="True"
            HorizontalOffset="40"
            Placement="Bottom"
            VerticalOffset="50">
            <TextBlock Background="LightBlue"
                Padding="3"
                Text="Popup Text"/>
        </Popup>
    </Canvas>
</Grid>
```

Результат приведенной выше разметки показан на рис. 33.



**Рис. 33.** Элемент управления Popup с указанием смещения точки выравнивания всплывающего окна

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.Controls.Popup.Offsets.CSharp](#)):

C#

```
var textBlock = new TextBlock
{
    Background = Brushes.LightBlue,
    Padding = new Thickness(3.0),
    Text = "Popup Text"
};
var popup = new Popup
{
    Child = textBlock,
    HorizontalOffset = 40.0,
    Placement = PlacementMode.Bottom,
    VerticalOffset = 50.0
};
var canvas = new Canvas
{
    Background = Brushes.Green,
    Margin = new Thickness(10.0, 10.0, 10.0, 100.0)
};
canvas.Children.Add(popup);
popup.IsOpen = true;
var grid = new Grid();
grid.Children.Add(canvas);
```

## 9.2. Взаимодействие свойств позиционирования между собой

Для того, чтобы определить корректную целевую область, целевую точку начала и точку смещения всплывающего окна, необходимо рассматривать свойства [System.Windows.Controls.Primitives.Popup.Placement](#), [System.Windows.Controls.Primitives.Popup.HorizontalOffset](#).

`Windows.Controls.Primitives.Popup.PlacementRectangle` и `System.Windows.Controls.Primitives.Popup.PlacementTarget` совместно.

Ниже приведено описание понятий, рассмотренных выше, для всех значений перечисления `System.Windows.Controls.Primitives.PlacementMode`:

Place- mentMode	Целевой объект	Целевая область	Целевая точка начала	Выравнивание всплывающе- го окна
Absolute	Не примени- мо. Свойство <code>PlacementTarget</code> игнорируется	Область экрана или значение свойства <code>PlacementRectangle</code> , если установлено.  <code>PlacementRectangle</code> описывает область относительно верхнего левого угла экрана.	Верхний левый угол целевой области	Верхний левый угол всплы- вающего окна
Absolute- Point	Не примени- мо. Свойство <code>PlacementTarget</code> игнорируется	Область экрана или значение свойства <code>PlacementRectangle</code> , если установлено.  <code>PlacementRectangle</code> описывает область относительно верхнего левого угла экрана.	Верхний левый угол целевой области	Верхний левый угол всплы- вающего окна
Bottom	Родительский элемент или значение свой- ства <code>Placement- Target</code> , если установлено	Целевой объект или значение свойства <code>PlacementRectangle</code> , если установлено.  <code>PlacementRectangle</code> описывают область относительно верхнего левого угла целевого объекта.	Нижний левый угол целевой области	Верхний левый угол всплы- вающего окна
Center	Родительский элемент или значение свой- ства <code>Placement- Target</code> , если установлено	Целевой объект или значение свойства <code>PlacementRectangle</code> , если установлено.  <code>PlacementRectangle</code> описывают область относительно верхнего левого угла целевого объекта.	Центр целевой области	Центр всплы- вающего окна

## Урок № 2

Place- mentMode	Целевой объект	Целевая область	Целевая точка начала	Выравнивание всплывающе- го окна
Custom	Родительский элемент или значение свойства PlacementTarget, если установлено	Целевой объект или значение свойства PlacementRectangle, если установлено. PlacementRectangle описывают область относительно верхнего левого угла целевого объекта.	Определяется при помощи CustomPopUpPlacementCallback	Определяется при помощи CustomPopUpPlacementCallback
Left	Родительский элемент или значение свойства PlacementTarget, если установлено	Целевой объект или значение свойства PlacementRectangle, если установлено. PlacementRectangle описывают область относительно верхнего левого угла целевого объекта	Верхний левый угол целевой области	Верхний правый угол всплывающего окна
Mouse	Не применимо. Свойство PlacementTarget игнорируется	Область курсора мыши. PlacementRectangle игнорируется	Нижний левый угол целевой области	Верхний левый угол всплывающего окна
Mouse-Point	Не применимо. Свойство PlacementTarget игнорируется.	Область курсора мыши. PlacementRectangle игнорируется	Верхний левый угол целевой области	Верхний левый угол всплывающего окна.
Relative	Родительский элемент или значение свойства PlacementTarget, если установлено	Целевой объект или значение свойства PlacementRectangle, если установлено. PlacementRectangle описывают область относительно верхнего левого угла целевого объекта	Верхний левый угол целевой области	Верхний левый угол всплывающего окна
Relative-Point	Родительский элемент или значение свойства PlacementTarget, если установлено	Целевой объект или значение свойства PlacementRectangle, если установлено. PlacementRectangle описывают область относительно верхнего левого угла целевого объекта	Верхний левый угол целевой области	Верхний левый угол всплывающего окна

PlacementMode	Целевой объект	Целевая область	Целевая точка начала	Выравнивание всплывающего окна
Right	Родительский элемент или значение свойства PlacementTarget, если установлено	Целевой объект или значение свойства PlacementRectangle, если установлено. PlacementRectangle описывают область относительно верхнего левого угла целевого объекта	Верхний правый угол целевой области	Верхний левый угол всплывающего окна
Top	Родительский элемент или значение свойства PlacementTarget, если установлено	Целевой объект или значение свойства PlacementRectangle, если установлено. PlacementRectangle описывают область относительно верхнего левого угла целевого объекта	Верхний левый угол целевой области	Нижний левый угол всплывающего окна

### 9.3. Перекрытие всплывающего окна границами экрана

Часть всплывающего окна не может быть перекрыта границей экрана. В случае, если подобная ситуация случается, то происходит одно из следующих действий:

- Всплывающее окно выравнивается по границе экрана, которая его перекрывает.
- Всплывающее окно использует другую точку выравнивания.
- Всплывающее окно использует другую точку выравнивания и другую целевую точку начала.

Точное поведение всплывающего окна при столкновении с одной из границ экрана зависит от значения, установленного для свойства `System.Windows.Controls.Primitives.Popup.Placement`.

Ниже приведено описание возможных видов столкновений с границами экрана, для всех значений перечисления System.Windows.Controls.Primitives.PlacementMode:

Placement-Mode	Верхняя граница	Нижняя граница	Левая граница	Правая граница
Absolute	Выравнивается по верхней границе экрана.	Выравнивается по нижней границе экрана.	Выравнивается по левой границе экрана.	Выравнивается по правой границе экрана.
Absolute-Point	Выравнивается по верхней границе экрана.	Точка выравнивания всплывающего окна смещается на нижний левый угол всплывающего окна.	Выравнивается по левой границе экрана.	Точка выравнивания всплывающего окна смещается на верхний правый угол всплывающего окна.
Bottom	Выравнивается по верхней границе экрана.	Целевая точка начала смещается на верхний левый угол целевой области, а точка выравнивания всплывающего окна смещается на нижний левый угол всплывающего окна.	Выравнивается по левой границе экрана.	Выравнивается по правой границе экрана.
Center	Выравнивается по верхней границе экрана.	Выравнивается по нижней границе экрана.	Выравнивается по левой границе экрана.	Выравнивается по правой границе экрана.
Left	Выравнивается по верхней границе экрана.	Выравнивается по нижней границе экрана.	Целевая точка начала смещается на верхний правый угол целевой области, а точка выравнивания всплывающего окна смещается на верхний левый угол всплывающего окна.	Выравнивается по правой границе экрана.
Mouse	Выравнивается по верхней границе экрана.	Целевая точка начала смещается на верхний левый угол целевой области, а	Выравнивается по левой границе экрана.	Выравнивается по правой границе экрана.

Placement-Mode	Верхняя граница	Нижняя граница	Левая граница	Правая граница
		точка выравнивания всплывающего окна смещается на нижний левый угол всплывающего окна.		
Mouse-Point	Выравнивается по верхней границе экрана.	Точка выравнивания всплывающего окна смещается на нижний левый угол всплывающего окна.	Выравнивается по левой границе экрана.	Точка выравнивания всплывающего окна смещается на верхний правый угол всплывающего окна.
Relative	Выравнивается по верхней границе экрана.	Выравнивается по нижней границе экрана.	Выравнивается по левой границе экрана.	Выравнивается по правой границе экрана.
Relative-Point	Выравнивается по верхней границе экрана.	Точка выравнивания всплывающего окна смещается на нижний левый угол всплывающего окна.	Выравнивается по левой границе экрана.	Точка выравнивания всплывающего окна смещается на верхний правый угол всплывающего окна.
Right	Выравнивается по верхней границе экрана.	Выравнивается по нижней границе экрана.	Выравнивается по левой границе экрана.	Целевая точка начала смещается на верхний левый угол целевой области, а точка выравнивания всплывающего окна смещается на верхний правый угол всплывающего окна.
Top	Целевая точка начала смещается на нижний левый угол целевой области, а точка выравнивания	Выравнивается по нижней границе экрана.	Выравнивается по левой границе экрана.	Выравнивается по правой границе экрана.

Placement-Mode	Верхняя граница	Нижняя граница	Левая граница	Правая граница
	всплывающего окна смещается на верхний левый угол всплывающего окна.			

## 9.4. Настройка позиционирования всплывающего окна

В WPF существует возможность настроить целевую точку начала и точку выравнивания всплывающего окна с учетом собственного алгоритма. Для этого необходимо установить свойству `System.Windows.Controls.Primitives.Popup.Placement` значение `System.Windows.Controls.Primitives.PlacementMode.Custom`, а также задать делегат, который возвращает набор допустимых точек позиционирования, установив значение свойству `System.Windows.Controls.Primitives.Popup.CustomPopupPlacementCallback`.

# 10. Меню

Меню в WPF являются вариацией списковых элементов управления (рис. 34). Если посмотреть на меню с точки зрения программиста, то можно понять, что меню является практически полной копией элемента управления, который отображает список элементов в иерархическом виде (`System.Windows.Controls.TreeView`).

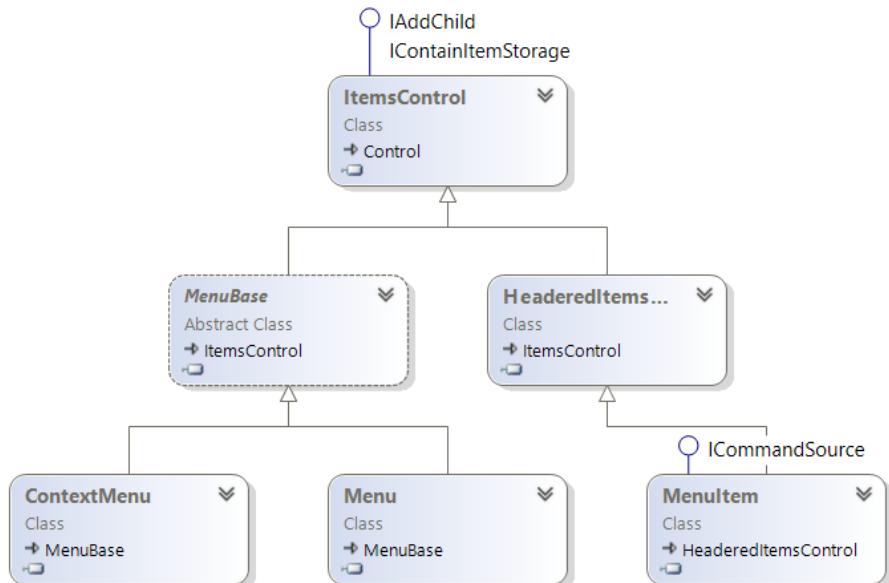


Рис. 34. Диаграмма классов, описывающих меню

## 10.1. Элемент управления `Menu`

Элемент управления `System.Windows.Controls.Menu` предназначен для отображения меню с иерархической структурой.

## Свойства класса System.Windows.Controls.Menu:

- **IsMainMenu** (System.Boolean). Получает или задает значение, которое указывает, должно ли текущее меню получать оповещения, адресованные главному меню окна, такие как нажатие клавиш *Alt* и *F10*. **True** если текущее меню должно получать оповещения, адресованные главному меню окна; иначе — **false**. Значение по умолчанию — **false**.

В качестве контейнера элементов для элемента управления System.Windows.Controls.Menu применяется класс System.Windows.Controls.MenuItem.

## Свойства класса System.Windows.Controls.MenuItem:

- **Icon** (System.Object). Получает или задает иконку пункта меню. Значение по умолчанию — **null**.
- **InputGestureText** (System.String). Получает или задает строку, описывающую жест устройства ввода, который активирует пункт меню, например, *Ctrl+C*. Значение по умолчанию — System.String.Empty.
- **IsCheckable** (System.Boolean). Получает или задает значение, указывающее может ли пункт меню быть включен/выключен. **True** если может быть включен/выключен; иначе — **false**. Значение по умолчанию — **false**.
- **.IsChecked** (System.Boolean). Получает или задает значение, которое указывает, включен пункт меню или нет. **True** если пункт меню включен; иначе — **false**. Значение по умолчанию — **false**.
- **IsHighlighted** (System.Boolean). Получает значение, которое указывает, подсвечен ли пункт меню. **True** если пункт меню подсвечен; иначе — **false**. Значение по умолчанию — **false**.

- **IsPressed** (System.Boolean). Получает значение, которое указывает, нажат ли пункт меню. **True** если пункт меню нажат; иначе — **false**. Значение по умолчанию — **false**.
- **IsSubMenuOpen** (System.Boolean). Получает или задает значение, которое указывает, открыто ли подменю пункта меню. **True** если подменю открыто; иначе — **false**. Значение по умолчанию — **false**.
- **Role** (System.Windows.Controls.MenuItemRole). Получает или задает роль пункта меню. Значение по умолчанию — System.Windows.Controls.MenuItemRole.TopLevelItem.
- **StaysOpenOnClick** (System.Boolean). Получает или задает значение, которое указывает, должно ли подменю, в котором находится текущий пункт меню, закрыться после нажатия на текущий пункту меню. **True** если подменю, содержащее текущий пункт меню не должно закрываться по нажатию на текущий пункт меню; иначе — **false**. Значение по умолчанию — **false**.

События класса System.Windows.Controls.MenuItem:

- **Checked** (System.Windows.RoutedEventHandler). Срабатывает, когда пункт меню включается.
- **Click** (System.Windows.RoutedEventHandler). Срабатывает, когда пункт меню нажимается.
- **SubmenuClosed** (System.Windows.RoutedEventHandler). Срабатывает, когда подменю закрывается.
- **SubmenuOpened** (System.Windows.RoutedEventHandler). Срабатывает, когда подменю открывается.
- **Unchecked** (System.Windows.RoutedEventHandler). Срабатывает, когда пункт меню выключается.

Для того, чтобы указать роль пункта меню необходимо использовать перечисление `System.Windows.Controls.MenuItemRole`, которое содержит следующие варианты:

- `SubmenuHeader`. Заголовок подменю.
- `SubmenuItem`. Пункт меню подменю (может выполнять команды).
- `TopLevelHeader`. Заголовок меню верхнего уровня.
- `TopLevelItem`. Пункт меню верхнего уровня (может выполнять команды).

При создании меню можно легко попасть в ситуацию, в которой какое-то подменю будет содержать слишком много пунктов меню. Для того, чтобы было удобнее взаимодействовать с таким меню, можно поделить пункты меня на логически связанные группы и отделить их друг от друга при помощи специального разделителя, описанного классом `System.Windows.Controls.Separator`.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке `Wpf.Controls.Menu.Xaml`):

### XAML

```
<Menu IsMainMenu="True">
    <MenuItem Header="File">
        <MenuItem Header="New">
            <MenuItem Header="Project..." InputGestureText="Ctrl+Shift+N"/>
            <MenuItem Header="File..." InputGestureText="Ctrl+N"/>
        </MenuItem>
        <MenuItem Header="Start Page"/>
        <Separator/>
        <MenuItem Header="Add">
```

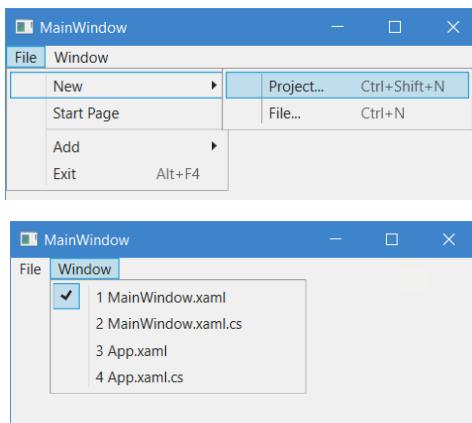
```

<MenuItem Header="New Project..."/>
<MenuItem Header="Existing Project..."/>
</MenuItem>
<MenuItem Header="Exit"
          InputGestureText="Alt+F4"/>
</MenuItem>

<MenuItem Header="Window">
    <MenuItem Header="1 MainWindow.xaml"
              IsCheckable="True"
              IsChecked="True"/>
    <MenuItem Header="2 MainWindow.xaml.cs"
              IsCheckable="True"/>
    <MenuItem Header="3 App.xaml"
              IsCheckable="True"/>
    <MenuItem Header="4 App.xaml.cs"
              IsCheckable="True"/>
</MenuItem>
</Menu>

```

Результат приведенной выше разметки показан на рис. 35.



**Рис. 35.** Элемент управления Menu

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.Menu.CSharp**):

**C#**

```
var separator = new Separator();

var menuItem31 = new MenuItem { Header = "Project...", 
    InputGestureText = "Ctrl+Shift+N" };
var menuItem32 = new MenuItem { Header = "File...", 
    InputGestureText = "Ctrl+N" };
var menuItem33 = new MenuItem { Header = "New 
    Project..." };
var menuItem34 = new MenuItem { Header = "Existing 
    Project..." };

var menuItem21 = new MenuItem { Header = "New" };
menuItem21.Items.Add(menuItem31);
menuItem21.Items.Add(menuItem32);
var menuItem22 = new MenuItem { Header = "Start Page" };
var menuItem23 = new MenuItem { Header = "Add" };
menuItem23.Items.Add(menuItem33);
menuItem23.Items.Add(menuItem34);
var menuItem24 = new MenuItem { Header = "Exit", 
    InputGestureText = "Alt+F4" };
var menuItem25 = new MenuItem
{
    Header = "1 MainWindow.xaml",
    IsCheckable = true,
    IsChecked = true
};
var menuItem26 = new MenuItem { Header =
    "2 MainWindow.xaml.cs", IsCheckable = true };
var menuItem27 = new MenuItem { Header =
    "3 App.xaml", IsCheckable = true };
var menuItem28 = new MenuItem { Header =
    "4 App.xaml.cs", IsCheckable = true };
```

```

var menuItem11 = new MenuItem { Header = "File" };
menuItem11.Items.Add(menuItem21);
menuItem11.Items.Add(menuItem22);
menuItem11.Items.Add(separator);
menuItem11.Items.Add(menuItem23);
menuItem11.Items.Add(menuItem24);
var menuItem12 = new MenuItem { Header = "Window" };
menuItem12.Items.Add(menuItem25);
menuItem12.Items.Add(menuItem26);
menuItem12.Items.Add(menuItem27);
menuItem12.Items.Add(menuItem28);

var menu = new Menu { IsMainMenu = true };
menu.Items.Add(menuItem11);
menu.Items.Add(menuItem12);

```

## 10.2. Элемент управления ContextMenu

Элемент управления System.Windows.Controls.ContextMenu предназначен для описания всплывающего меню, которое позволяет элементам управления предоставлять дополнительную функциональность в зависимости от контекста.

Свойства класса System.Windows.Controls.ContextMenu:

- **CustomPopupPlacementCallback** (System.Windows.Controls.Primitives.CustomPopupPlacementCallback). *Получает или задает метод обратного вызова, который используется для определения позиционирования контекстного меню, если значение свойства System.Windows.Controls.ContextMenu.Placement равно System.Windows.Controls.Primitives.PlacementMode.Custom.*

- **HasDropShadow** (System.Boolean). Получает или задает значение, которое указывает, должно ли контекстное меню отбрасывать тень. **True** если контекстное меню должно отбрасывать тень; иначе — **false**. Значение по умолчанию — **false**.
- **HorizontalOffset** (System.Double). Получает или задает горизонтальный отступ относительно целевой области. Значение по умолчанию — 0.0.
- **IsOpen** (System.Boolean). Получает или задает значение, которое указывает, отображается ли контекстное меню. **True** если контекстное меню отображается; иначе — **false**. Значение по умолчанию — **false**.
- **Placement** (System.Windows.Controls.Primitives.PlacementMode). Получает или задает режим расположения контекстного меню. Значение по умолчанию — System.Windows.Controls.Primitives.PlacementMode.MousePoint.
- **PlacementRectangle** (System.Windows.Rect). Получает или задает область, относительно которой позиционируется контекстное меню при отображении. Значение по умолчанию — System.Windows.Rect.Empty.
- **PlacementTarget** (System.Windows.UIElement). Получает или задает элемент, относительно которого позиционируется контекстное меню при отображении. Значение по умолчанию — **null**.
- **StaysOpen** (System.Boolean). Получает или задает значение, которое указывает, должно ли меню закрываться автоматически. **True** если меню должно оставаться открытым до тех пор, пока данному свойству не будет

присвоено значение `false`; иначе — `false`. Значение по умолчанию — `false`.

- **VerticalOffset** (`System.Double`). Получает или задает вертикальный отступ относительно целевой области. Значение по умолчанию — 0.0.

События класса `System.Windows.Controls.ContextMenu`:

- **Closed** (`System.Windows.RoutedEventArgs`). Срабатывает, когда контекстное меню закрывается.
- **Opened** (`System.Windows.RoutedEventArgs`). Срабатывает, когда контекстное меню открывается.

В качестве контейнера элементов для элемента управления `System.Windows.Controls.ContextMenu` применяется класс `System.Windows.Controls.MenuItem`.

Контекстное меню может быть установлено любому элементу, тип которого является производным от `System.Windows.Controls.FrameworkElement`, так как именно в нем объявлено свойство `System.Windows.Controls.FrameworkElement.ContextMenu`.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке `Wpf.Controls.ContextMenu.Xaml`):

### XAML

```
<ContextMenu>
    <MenuItem Header="Edit">
        <MenuItem Header="Cut"
                  InputGestureText="Ctrl+X"/>
        <MenuItem Header="Copy"
                  InputGestureText="Ctrl+C"/>
        <MenuItem Header="Paste"
                  InputGestureText="Ctrl+V"/>
```

```
</MenuItem>
<MenuItem Header="Select All"
          InputGestureText="Ctrl+A"/>
</ContextMenu>
```

Результат приведенной выше разметки показан на рис. 36.

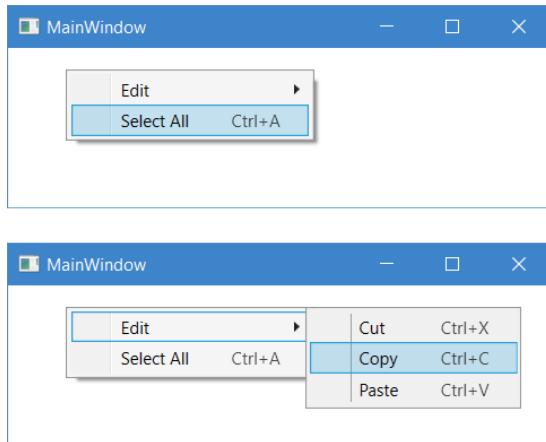


Рис. 36. Элемент управления ContextMenu

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.ContextMenu.CSharp**):

C#

```
var menuItem21 = new MenuItem { Header = "Cut",
    InputGestureText = "Ctrl+X" };
var menuItem22 = new MenuItem { Header = "Copy",
    InputGestureText = "Ctrl+C" };
var menuItem23 = new MenuItem { Header = "Paste",
    InputGestureText = "Ctrl+V" };
var menuItem11 = new MenuItem { Header = "Edit" };
```

```
menuItem11.Items.Add(menuItem21);
menuItem11.Items.Add(menuItem22);
menuItem11.Items.Add(menuItem23);
var menuItem12 = new MenuItem { Header =
    "Select All", InputGestureText = "Ctrl+A" };

var contextMenu = new ContextMenu();
contextMenu.Items.Add(menuItem11);
contextMenu.Items.Add(menuItem12);
```

# 11. Подсказки

Элемент управления `System.Windows.Controls.ToolTip` предназначен для отображения подсказок пользователю касательно определенного элемента управления или части интерфейса приложения.

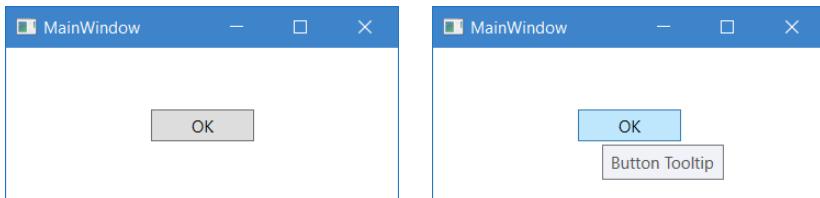
Для того, чтобы установить подсказку элементу управления, необходимо использовать его свойство `System.Windows.FrameworkElement.ToolTip`.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке `Wpf.Controls.ToolTip.Simple.Xaml`):

## XAML

```
<Button Height="23" ToolTip="Button Tooltip"  
       Width="75">OK</Button>
```

Результат приведенной выше разметки показан на рис. 37.



**Рис. 37.** Элемент управления `Button`, обладающий простой подсказкой

Рассмотренной выше разметке соответствует следующий код (полный пример находится в `Wpf.Controls.ToolTip.Simple.CSharp`):

**C#**

```
var button = new Button
{
    Content = "OK",
    Height = 23.0,
    ToolTip = "Button Tooltip",
    Width = 75.0
};
```

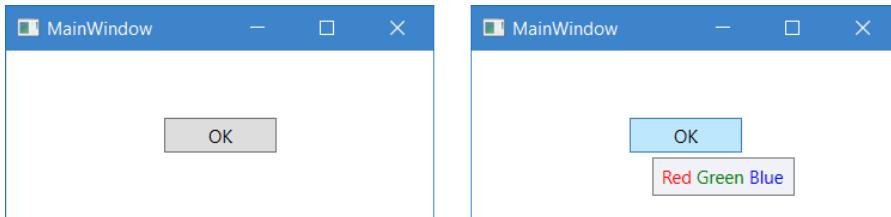
Свойство `System.Windows.FrameworkElement.ToolTip` обладает типом `System.Object`, что позволяет записать в него любой объект. Все, что будет установлено в качестве значения, будет использовано как содержимое для всплывающего окна, в котором располагается подсказка. Это позволяет описать практически любую структуру подсказки.

Приведенный ниже фрагмент разметки демонстрирует подсказку с более сложной структурой (полный пример находится в папке `Wpf.Controls.ToolTip.Advanced.Xaml`):

**XAML**

```
<Button Content="OK" Height="23" Width="75">
    <Button.ToolTip>
        <StackPanel Orientation="Horizontal">
            <TextBlock Foreground="Red" Text="Red ">/</TextBlock>
            <TextBlock Foreground="Green" Text="Green ">/</TextBlock>
            <TextBlock Foreground="Blue" Text="Blue"/>
        </StackPanel>
    </Button.ToolTip>
</Button>
```

Результат приведенной выше разметки показан на рис. 38.



**Рис. 38.** Элемент управления Button, обладающий более сложной подсказкой

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.ToolTip.Advanced.CSharp**):

C#

```
var textBlock1 = new TextBlock { Foreground =
    Brushes.Red, Text = "Red " };
var textBlock2 = new TextBlock { Foreground =
    Brushes.Green, Text = "Green " };
var textBlock3 = new TextBlock { Foreground =
    Brushes.Blue, Text = "Blue" };
var stackPanel = new StackPanel { Orientation =
    Orientation.Horizontal };
stackPanel.Children.Add(textBlock1);
stackPanel.Children.Add(textBlock2);
stackPanel.Children.Add(textBlock3);

var button = new Button
{
    Content = "OK",
    Height = 23.0,
    ToolTip = stackPanel,
    Width = 75.0
};
```

Для настройки разнообразных параметров подсказки, необходимо использовать присоединенные свойства или

события класса `System.Windows.Controls.ToolTipService` на элементе, требующем настройки.

Присоединенные свойства класса `System.Windows.Controls.ToolTipService`:

- **BetweenShowDelay** (`System.Int32`). Получает или задает максимальное количество миллисекунд между показами двух подсказок, в случае, когда вторая подсказка показывается без начальной задержки. Значение по умолчанию — 100.0.
- **HasDropShadow** (`System.Boolean`). Получает или задает значение, которое указывает, должна ли подсказка отбрасывать тень. `True` если подсказка должна отбрасывать тень; иначе — `false`. Значение по умолчанию — `false`.
- **HorizontalOffset** (`System.Double`). Получает или задает горизонтальный отступ относительно целевой области. Значение по умолчанию — 0.0.
- **InitialShowDelay** (`System.Int32`). Получает или задает количество миллисекунд, которое должно пройти перед тем, как показать подсказку. Значение по умолчанию — `System.Windows.SystemParameters.MouseHoverTime`.
- **.IsEnabled** (`System.Boolean`). Получает или задает значение, которое указывает, необходимо ли показывать подсказку. `True` если подсказку необходимо показывать; иначе — `false`. Значение по умолчанию — `true`.
- **IsOpen** (`System.Boolean`). Получает или задает значение, которое указывает, отображается ли подсказка. `True` если подсказка отображается; иначе — `false`. Значение по умолчанию — `false`.
- **Placement** (`System.Windows.Controls.Primitives.PlacementMode`). Получает или задает режим распо-

локации подсказки. Значение по умолчанию — System.Windows.Controls.Primitives.[PlacementMode](#).Mouse.

- **PlacementRectangle** (System.Windows.Rect). Получает или задает область, относительно которой позиционируется подсказка при отображении. Значение по умолчанию — System.Windows.Rect.Empty.
- **PlacementTarget** (System.Windows.UIElement). Получает или задает элемент, относительно которого позиционируется подсказка при отображении. Значение по умолчанию — [null](#).
- **ShowDuration** (System.Int32). Получает или задает количество миллисекунд, которое подсказка остается видимой. Значение по умолчанию — 5000.0.
- **ShowOnDisabled** (System.Boolean). Получает или задает значение, которое указывает, необходимо ли показывать подсказку, если элемент, которому она принадлежит отключен. [True](#) если подсказка должна быть показана даже если элемент отключен; иначе — [false](#). Значение по умолчанию — [false](#).
- **VerticalOffset** (System.Double). Получает или задает вертикальный отступ относительно целевой области. Значение по умолчанию — 0.0.

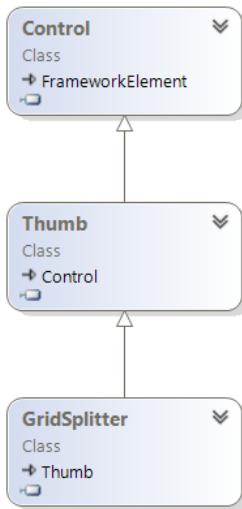
Присоединенные события класса System.Windows.Controls.[ToolTipService](#):

- **ToolTipClosing** (System.Windows.Controls.ToolTipEventArgs). Срабатывает, когда подсказка закрывается.
- **ToolTipOpening** (System.Windows.Controls.ToolTipEventArgs). Срабатывает, когда подсказка открывается.

Обычно, когда пользователь перемещает курсор мыши на область над элементом управления, который обладает подсказкой, то должно пройти некоторое время перед тем, как подсказка будет показана. Значение такой первоначальной задержки описывается свойством `System.Windows.Controls.ToolTipService.InitialShowDelay`. Однако, после того, как подсказка была показана существует отрезок времени, в котором следующая подсказка может быть показана без начальной задержки. Этот промежуток времени указывается свойством `System.Windows.Controls.ToolTipService.BetweenShowDelay`. Когда пользователь перемещает курсор мыши в пределах этого временного отрезка с одного элемента, обладающего открытой подсказкой, на другой, который также способен отображать подсказки, то вторая подсказка отображается сразу же, без начальной задержки.

# 12. Перетаскиваемые элементы управления

В WPF существуют элементы управления, основной задачей которых является предоставление функциональности «перетаскивания» пользователю. Базовым классом для них является класс `System.Windows.Controls.Primitives.Thumb` (рис. 39).



**Рис. 39.** Диаграмма классов, описывающих перетаскиваемые элементы управления

Свойства класса `System.Windows.Controls.Primitives.Thumb`:

- **IsDragging** (`System.Boolean`). Получает значение, которое указывает, обладает ли элемент логическим фокусом, захватом мыши и нажата ли левая кнопка мыши. `True`

если элемент обладает логическим фокусом, захватом мыши и нажата левая кнопка мыши; иначе — **false**. Значение по умолчанию — **false**.

События класса System.Windows.Controls.Primitives.

### Thumb:

- **DragCompleted** (System.Windows.Controls.Primitives.DragCompletedEventArgs). Срабатывает, когда элемент теряет захват мыши.
- **DragDelta** (System.Windows.Controls.Primitives.DragDeltaEventArgs). Срабатывает, когда смещается курсор мыши, в то время как элемент обладает логическим фокусом и захватом мыши.
- **DragStarted** (System.Windows.Controls.Primitives.DragStartedEventArgs). Срабатывает, когда элемент получает логический фокус и захват мыши.

Методы класса System.Windows.Controls.Primitives.

### Thumb:

- **CancelDrag()**. Отменяет операцию перемещения.

## 12.1. Элемент управления GridSplitter

Элемент управления System.Windows.Controls.GridSplitter предназначен для того, чтобы перераспределять пространство между строками или колонками панели System.Windows.Controls.Grid.

Свойства класса System.Windows.Controls.GridSplitter:

- **DragIncrement** (System.Double). Получает или задает минимальное расстояние, которое необходимо преодолеть курсору мыши, чтобы изменить размеры строк или колонок. Значение по умолчанию — 1.0.

- **KeyboardIncrement** (System.Double). Получает или задает величину сдвига для каждого нажатия клавиш стрелок. Значение по умолчанию — 10.0.
- **ResizeBehavior** (System.Windows.Controls.GridResizeBehavior). Получает или задает, какие именно строки или колонки по отношению к той, в которой находится элемент, будут менять свои размеры. Значение по умолчанию — System.Windows.Controls.GridResizeBehavior.BasedOnAlignment.
- **ResizeDirection** (System.Windows.Controls.GridResizeDirection). Получает или задает размер чего именно (строк или колонок) будет меняться при помощи элемента. Значение по умолчанию — System.Windows.Controls.GridResizeDirection.Auto.
- **ShowsPreview** (System.Boolean). Получает или задает значение, которое указывает, необходимо ли обновлять размеры строк или колонок во время перетаскивания. **True** если размеры строк или колонок необходимо обновлять во время перетаскивания; иначе — **false**. Значение по умолчанию — **false**.

Для того, чтобы указать поведение при перетаскивании элемента необходимо использовать перечисление System.Windows.Controls.GridResizeBehavior, которое содержит следующие варианты:

- **BasedOnAlignment**. Пространство распределяется на основании значения свойств System.Windows.FrameworkElement.HorizontalAlignment и System.Windows.FrameworkElement.VerticalAlignment.
- **CurrentAndNext**. Если элемент ориентирован горизонтально, то пространство распределяется между строкой,

в которой он помещен и следующей строкой (той, что снизу). Если элемент ориентирован вертикально, то пространство распределяется между колонкой, в которой он помещен и следующей колонкой (той, что справа).

- PreviousAndCurrent. Если элемент ориентирован горизонтально, то пространство распределяется между строкой, в которой он помещен и предыдущей строкой (той, что сверху). Если элемент ориентирован вертикально, то пространство распределяется между колонкой, в которой он помещен и предыдущей колонкой (той, что слева).
- PreviousAndNext. Если элемент ориентирован горизонтально, то пространство распределяется между предыдущей и следующей строкой. Если элемент ориентирован вертикально, то пространство распределяется между предыдущей и следующей колонкой.

Для того, чтобы указать направление перетаскивания элемента необходимо использовать перечисление System.Windows.Controls.GridResizeDirection, которое содержит следующие варианты:

- Auto. Пространство распределяется на основании значения свойств System.Windows.FrameworkElement.HorizontalAlignment, System.Windows.FrameworkElement.VerticalAlignment, System.Windows.FrameworkElement.ActualHeight и System.Windows.FrameworkElement.ActualWidth элемента.
- Columns. Пространство распределяется между колонками.
- Rows. Пространство распределяется между строками.

Обычно, при использовании элемента, распределяющего пространство между строками или колонками, выделяют под него отдельную строку или колонку соответственно.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке **Wpf.Controls.GridSplitter.Xaml**):

### XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition Height="Auto"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Border Background="Aqua"
            Grid.Column="0"
            Grid.Row="0"/>
    <Border Background="Brown"
            Grid.Column="2"
            Grid.Row="0"/>
    <Border Background="Bisque"
            Grid.Column="3"
            Grid.Row="0"/>
    <Border Background="DarkViolet"
            Grid.Column="0"
            Grid.Row="2"/>
    <Border Background="Khaki"
            Grid.Column="2"
            Grid.Row="2"/>
```

```

<Border Background="Orange"
        Grid.Column="3"
        Grid.Row="2"/>
<GridSplitter Grid.Column="1"
              Grid.Row="0"
              Grid.RowSpan="3"
              HorizontalAlignment="Stretch"
              Width="3"/>
<GridSplitter Height="3"
              Grid.Column="0"
              Grid.ColumnSpan="4"
              Grid.Row="1"
              HorizontalAlignment="Stretch"/>
</Grid>

```

Результат приведенной выше разметки показан на рис. 40.



**Рис. 40.** Элемент управления GridSplitter

Рассмотренной выше разметке соответствует следующий код (полный пример находится в **Wpf.Controls.GridSplitter.CSharp**):

C#

```

var border1 = new Border { Background = Brushes.Aqua };
var border2 = new Border { Background = Brushes.Brown };

```

```
var border3 = new Border { Background = Brushes.Bisque };
var border4 = new Border { Background =
    Brushes.DarkViolet };
var border5 = new Border { Background = Brushes.Khaki };
var border6 = new Border { Background = Brushes.Orange};

var gridSplitter1 = new GridSplitter
{
    HorizontalAlignment = HorizontalAlignment.Stretch,
    Width = 3.0
};
var gridSplitter2 = new GridSplitter
{
    Height = 3.0,
    HorizontalAlignment = HorizontalAlignment.Stretch
};

var grid = new Grid();
grid.ColumnDefinitions.Add(new ColumnDefinition());
grid.ColumnDefinitions.Add(new ColumnDefinition { Width =
    GridLength.Auto });
grid.ColumnDefinitions.Add(new ColumnDefinition());
grid.ColumnDefinitions.Add(new ColumnDefinition());
grid.RowDefinitions.Add(new RowDefinition());
grid.RowDefinitions.Add(new RowDefinition { Height =
    GridLength.Auto });
grid.RowDefinitions.Add(new RowDefinition());

grid.Children.Add(border1);
grid.Children.Add(border2);
grid.Children.Add(border3);
grid.Children.Add(border4);
grid.Children.Add(border5);
grid.Children.Add(border6);
grid.Children.Add(gridSplitter1);
grid.Children.Add(gridSplitter2);
```

```
Grid.SetColumn(border1, 0);
Grid.SetRow(border1, 0);
Grid.SetColumn(border2, 2);
Grid.SetRow(border2, 0);
Grid.SetColumn(border3, 3);
Grid.SetRow(border3, 0);
Grid.SetColumn(border4, 0);
Grid.SetRow(border4, 2);
Grid.SetColumn(border5, 2);
Grid.SetRow(border5, 2);
Grid.SetColumn(border6, 3);
Grid.SetRow(border6, 2);
Grid.SetColumn(gridSplitter1, 1);
Grid.SetRow(gridSplitter1, 0);
Grid.SetRowSpan(gridSplitter1, 3);
Grid.SetColumn(gridSplitter2, 0);
Grid.SetColumnSpan(gridSplitter2, 4);
Grid.SetRow(gridSplitter2, 1);
```

# 13. Маршрутизация событий

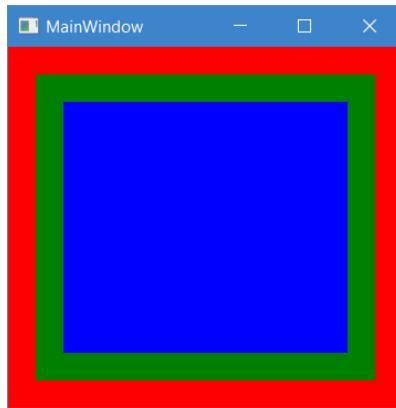
Как было сказано ранее, в WPF существует два вида маршрутизации событий: туннелирование и всплытие. Маршрутизируемые события, обычно, представляются парами: предварительное событие, которое туннелируется и основное событие, которое всплывает. Предварительное события, как правило, называется так же, как и основное, с которым оно в паре, но при этом обладает дополнительным префиксом — Preview. Например, существует основное событие System.Windows.UIElement.MouseDown и его предварительная версия System.Windows.UIElement.PreviewMouseDown.

Для лучшего понимания процесса туннелирования и всплытия, давайте рассмотрим следующий фрагмент разметки (полный пример находится в папке **Wpf.Events.Routing.Xaml**):

## XAML

```
<Window Height="300" Width="300">
    <Grid Background="Red">
        <Border Background="Green" Margin="20">
            <Rectangle Fill="Blue" Margin="20"/>
        </Border>
    </Grid>
</Window>
```

Результат приведенной выше разметки показан на рис. 41.



**Рис. 41.** Туннелирование и всплытие события

Если нажать кнопку мыши на синем прямоугольнике (элемент `<Rectangle>`), то произойдет следующее. Сначала будет сгенерировано предварительное событие описывающее нажатие кнопки мыши — `System.Windows.UIElement.PreviewMouseDown`. Данное событие будет туннелироваться, т.е. будет направлено от корневого элемента (`<Window>`) до того, кому оно адресовано (источник события). В этом случае предварительное событие нажатия кнопки мыши пройдет элементы в следующем порядке: `<Window>, <Grid>, <Border>, <Rectangle>`. После этого будет сгенерировано собственно событие нажатия кнопки мыши — `System.Windows.UIElement.MouseDown`, которое будет вспыльвать по визуальному дереву элементов, начиная с элемента-источника в следующем порядке: `<Rectangle>, <Border>, <Grid>, <Window>`.

### 13.1. Прямая маршрутизация

В действительности существует еще и третий вид маршрутизации — прямая маршрутизация. На самом деле,

в этом случае не происходит никакой маршрутизации, так как событие генерируется только для элемента-источника и не перемещается по визуальному дереву элементов. Такие «прямые» события необходимы в ситуациях, когда всплытие события не имеет смысла. Например, событие перемещения курсора мыши на область нового элемента — `System.Windows.UIElement.MouseEnter`. Если прикрепить обработчик данного события на уровне окна, то вполне оправдано будет ожидать срабатывание данного обработчика каждый раз, как курсор мыши перемещается на область окна извне. Именно это и происходит в действительности. Однако, если бы такое событие было маршрутизируемым и всплывало бы, то каждый раз, при перемещении курсора мыши в пределах окна с одного элемента на другой, оповещение получал бы и элемент, и окно. Более того, данное событие срабатывало бы для всех элементов в цепочке родительских элементов для того, на который переместился курсор мыши. А это не то, что ожидается от события такого рода. По этой причине описанное событие использует прямую маршрутизацию.

## 13.2. Проверка на визуальное попадание

Когда пользователь выполняет какие-либо манипуляции с мышью, приложение должно определить какому именно элементу должно быть адресовано соответствующее совершенным действиям событие. Элемент находящийся непосредственно под курсором мыши является целевым, так называемый источник события. Для этого производится обход визуального дерева элементов, с учетом координат курсора. Однако есть ряд правил, кото-

рые могут изменить результат этого процесса, а именно, сделать элемент «невидимым» для подобной проверки. Элементы визуального дерева обладают свойством System.Windows.UIElement.IsHitTestVisible, которое отвечает за то, учитывается ли элемент (и его дочерние элементы) в алгоритме проверки на визуальное попадание (hit test).

В приведенном ниже фрагменте разметки элемент **<Border>** помечен, как невидимый для проверки на визуальное попадание (полный пример находится в папке **Wpf.Properties.IsHitTestVisible.Xaml**):

### XAML

```
<Window Height="300" Width="300">
    <Grid Background="Red">
        <Border Background="Green"
            IsHitTestVisible="False"
            Margin="20">
            <Rectangle Fill="Blue" Margin="20"/>
        </Border>
    </Grid>
</Window>
```

Если нажать кнопку мыши на синем прямоугольнике (элемент **<Rectangle>**), то произойдет следующее. Сначала произойдет туннелирование события System.Windows.UIElement.PreviewMouseDown, в следующем порядке: **<Window>, <Grid>**. Элемент **<Border>** и его дочерние не будут учитываться. После этого произойдет всплытие события System.Windows.UIElement.MouseDown, в следующем порядке: **<Grid>, <Window>**. Фактически, произошло срабатывание события так, как будто элемент **<Border>** вообще отсутствует в визуальном дереве.

Подобного эффекта можно добиться и другим способом, обычно, не желая этого, даже если свойство `System.Windows.UIElement.IsHitTestVisible` будет указывать, что элемент должен учитываться при проверке на визуальное попадание. Если не задать элементу кисть для рисования заднего фона, то прозрачная область клиентской области элемента не будет учитываться. Но если при этом у элемента будет рамка не нулевой толщины с установленной кистью, то область рамки будет учитываться. Если все же необходимо, чтобы элемент был прозрачным, но при этом учитывался при проверке на визуальное попадание, то необходимо указать прозрачную кисть. Другими словами, чтобы элемент учитывался, свойство описывающее кисть заднего фона, должно содержать значение отличное от `null`. Прозрачную кисть можно задать одним из следующих способов:

### XAML

```
<Border Background="Transparent"
        Height="100"
        Width="100"/>
```

### C#

```
var border = new Border
{
    Background = Brushes.Transparent,
    Height = 100.0,
    Width = 100.0
};
```

Свойства класса `System.Windows.UIElement`, связанные с проверкой на визуальное попадание:

- **IsHitTestVisible** (System.Boolean). Получает или задает значение, которое указывает может ли элемент быть возвращен в качестве результата операции проверки на визуальное попадание курсора на элемент. **True**, если может; иначе — **false**. Значение по умолчанию — **true**.

События класса System.Windows.UIElement, связанные с проверкой на визуальное попадание:

- **IsHitTestVisibleChanged** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement.IsHitTestVisible.

Методы класса System.Windows.UIElement, связанные с проверкой на визуальное попадание:

- **InputHitTest(point)**. Возвращает элемент, находящийся по указанным координатам, относительно текущего элемента.

### 13.3. Параметры маршрутизируемых событий

Создавая события в .NET Framework, принято использовать для них делегаты определенного вида. Такие делегаты должны описывать методы, которые ничего не возвращают, и принимают два параметра: первый параметр описывает объект, к которому прикреплен обработчик события и для которого он вызывается, а второй хранит дополнительную информацию, специфичную для события (например, координаты курсора мыши или код нажатой клавиши). В качестве типа данных для второго параметра принято использовать класс System.EventArgs, если событие не требует дополнительной информации,

и производный от него класс, если дополнительная информацию требуется. Для маршрутизируемых событий в WPF существует промежуточный класс — System.Windows.RoutedEventArgs, который в свою очередь наследуется от System.EventArgs. Это необходимо, потому что, даже если маршрутизуемое событие не требует никакой дополнительной информации специфичной для него, оно должно хранить информацию о маршрутизации.

Свойства класса System.Windows.RoutedEventArgs:

- **Handled** (System.Boolean). Получает или задает значение, которое указывает текущее состояние маршрутизуемого события. **True** если событие обработано; иначе — **false**. Значение по умолчанию — **false**.
- **OriginalSource** (System.Object). Получает объект-источник события. В данном случае учитываются элементы из визуального дерева, т.е. если событие сработает на какой-то составляющей части, например, поля для ввода текста, то данное свойство будет содержать именно эту часть.
- **RoutedEvent** (System.Windows.RoutedEventArgs). Получает или задает объект, ассоциированный с данным событием.
- **Source** (System.Object). Получает или задает объект-источник события. В данном случае учитываются элементы из логического дерева, т.е. если событие сработает на какой-то составляющей части, например, поля для ввода текста, то данное свойство будет содержать поле для ввода текста.

Делегаты, применяющиеся для маршрутизируемых событий, устроены по одному принципу, меняется лишь тип второго параметра:

C#

```
delegate void RoutedEventHandler(object sender,  
RoutedEventArgs e);
```

# 14. Обработка ввода

В данном разделе будет рассмотрен API предоставляемый WPF для получения ввода с различных устройств, к наиболее распространенным из которых можно отнести клавиатуру и мышь. Большая часть функциональности связанной с обработкой ввода находится в базовом классе `System.Windows.UIElement`.

## 14.1. Мышь

Обработка ввода при помощи мыши в большинстве случаев сводится к обработке требуемых событий мыши, которые происходят при нажатии и отпускании кнопок, перемещении курсора и прокрутке колеса.

Свойства класса `System.Windows.UIElement`, связанные с мышью:

- **IsMouseCaptured** (`System.Boolean`). Получает значение, которое указывает, захвачена ли мышь элементом. `True`, если мышь захвачена элементом; иначе — `false`.
- **IsMouseCaptureWithin** (`System.Boolean`). Получает значение, которое указывает, захвачена ли мышь элементом или одним из дочерних элементов. `True`, если мышь захвачена элементом или одним из дочерних элементов; иначе — `false`.
- **IsMouseDirectlyOver** (`System.Boolean`). Получает значение, которое указывает, находится ли курсор мыши над элементом (не включая дочерние элементы в визуальном дереве). `True`, если курсор мыши находится над элементом; иначе — `false`.

- **IsMouseOver** (System.Boolean). Получает значение, которое указывает, находится ли курсор мыши над элементом (включая дочерние элементы в визуальном дереве). **True**, если курсор мыши находится над элементом; иначе — **false**.

События класса System.Windows.UIElement, связанные с мышью:

- **GotMouseCapture** (System.Windows.Input.MouseEventHandler). Срабатывает, когда элемент захватывает мышь.
- **IsMouseCapturedChanged** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement.IsMouseCaptured.
- **IsMouseCaptureChangedWithin** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement.IsMouseCaptureWithin.
- **IsMouseDirectlyOverChanged** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement.IsMouseDirectlyOver.
- **LostMouseCapture** (System.Windows.Input.MouseEventHandler). Срабатывает, когда элемент отпускает захват мыши.
- **MouseDown** (System.Windows.Input.MouseEventHandler). Срабатывает, при нажатии любой из кнопок мыши, пока курсор мыши находится над элементом.
- **MouseEnter** (System.Windows.Input.MouseEventHandler). Событие срабатывает, когда курсор мыши переме-

щается на область элемента с другого элемента или извне.

- **MouseLeave** ([System.Windows.Input.MouseEventHandler](#)). Срабатывает, когда курсор мыши покидает область элемента.
- **MouseLeftButtonDown** ([System.Windows.Input.MouseEventHandler](#)). Срабатывает, когда нажимается левая кнопка мыши, пока курсор мыши находится над элементом.
- **MouseLeftButtonUp** ([System.Windows.Input.MouseEventHandler](#)). Срабатывает, когда отпускается левая кнопка мыши, пока курсор мыши находится над элементом.
- **MouseMove** ([System.Windows.Input.MouseEventHandler](#)). Срабатывает, когда курсор мыши перемещается над областью элемента.
- **MouseRightButtonDown** ([System.Windows.Input.MouseEventHandler](#)). Срабатывает, при нажатии правой кнопки мыши, пока курсор находится над элементом.
- **MouseRightButtonUp** ([System.Windows.Input.MouseEventHandler](#)). Срабатывает, когда отпускается правая кнопка мыши, пока курсор мыши находится над элементом.
- **MouseUp** ([System.Windows.Input.MouseEventHandler](#)). Срабатывает, при отпускании любой из кнопок мыши, пока курсор мыши находится над элементом.
- **MouseWheel** ([System.Windows.Input.MouseEventHandler](#)). Срабатывает, когда прокручивается колесо мыши, пока курсор мыши находится над элементом.

- **PreviewMouseDown** (`System.Windows.Input.MouseEventHandler`). Предварительное событие для события `System.Windows.UIElement.MouseDown`.
- **PreviewMouseLeftButtonDown** (`System.Windows.Input.MouseEventHandler`). Предварительное событие для события `System.Windows.UIElement.MouseLeftButtonDown`.
- **PreviewMouseLeftButtonUp** (`System.Windows.Input.MouseEventHandler`). Предварительное событие для события `System.Windows.UIElement.MouseLeftButtonUp`.
- **PreviewMouseMove** (`System.Windows.Input.MouseEventHandler`). Предварительное событие для события `System.Windows.UIElement.MouseMove`.
- **PreviewMouseRightButtonDown** (`System.Windows.Input.MouseEventHandler`). Предварительное событие для события `System.Windows.UIElement.MouseRightButtonDown`.
- **PreviewMouseRightButtonUp** (`System.Windows.Input.MouseEventHandler`). Предварительное событие для события `System.Windows.UIElement.MouseRightButtonUp`.
- **PreviewMouseUp** (`System.Windows.Input.MouseEventHandler`). Предварительное событие для события `System.Windows.UIElement.MouseUp`.
- **PreviewMouseWheel** (`System.Windows.Input.MouseWheelEventHandler`). Предварительное событие для события `System.Windows.UIElement.MouseWheel`.

Методы класса `System.Windows.UIElement`, связанные с мышью:

- **CaptureMouse()**. Пытается захватить мышь. Возвращает `true`, если операция выполнилась успешно; иначе — `false`.
- **ReleaseMouseCapture()**. Отпускает захват мыши, если текущий элемент захватывал ее.

События класс `System.Windows.Controls.Control`, связанные с мышью:

- **MouseDoubleClick** (`System.Windows.Input.MouseEventHandler`). Срабатывает, когда производится двойной щелчок кнопкой мыши, пока курсор мыши находится над элементом.
- **PreviewMouseDoubleClick** (`System.Windows.Input.MouseEventHandler`). Предварительное событие для события `System.Windows.Controls.Control.MouseDoubleClick`.

В дополнение к API для взаимодействия с мышью, помещенному в базовые классы элементов, существует статический класс `System.Windows.Input.Mouse`, представляющий набор полезных свойств и методов.

Статические свойства класса `System.Windows.Input.Mouse`:

- **Captured** (`System.Windows.IInputElement`). Получает элемент, который захватил мышь в текущий момент.
- **DirectlyOver** (`System.Windows.IInputElement`). Получает элемент, над которым находится курсор мыши.
- **LeftButton** (`System.Windows.Input.MouseButtonState`). Получает состояние левой кнопки мыши.

- **MiddleButton** (System.Windows.Input.MouseButtonState).  
Получает состояние средней кнопки мыши.
- **RightButton** (System.Windows.Input.MouseButtonState).  
Получает состояние правой кнопки мыши.
- **XButton1** (System.Windows.Input.MouseButtonState).  
Получает состояние первой дополнительной кнопки мыши.
- **XButton2** (System.Windows.Input.MouseButtonState).  
Получает состояние второй дополнительной кнопки мыши.

Статические методы класса System.Windows.Input.Mouse:

- **GetPosition(relativeTo)**. Возвращает позицию курсора мыши относительно указанного элемента.
- **SetCursor(cursor)**. Устанавливает указанный курсор мыши.

Для того, чтобы узнать состояние кнопки мыши необходимо использовать перечисление System.Windows.Input.MouseButtonState, которое содержит следующие варианты:

- Pressed. Кнопка нажата.
- Released. Кнопка отпущена.

## 14.2. Клавиатура

События класса System.Windows.UIElement, связанные с клавиатурой:

- **KeyDown** (System.Windows.Input.KeyEventHandler).  
Срабатывает, когда нажимается клавиша на клавиатуре, пока элемент обладает клавиатурным фокусом.

- **KeyUp** (System.Windows.Input.KeyEventHandler). Срабатывает, когда отпускается клавиша на клавиатуре, пока элемент обладает клавиатурным фокусом.
- **PreviewKeyDown** (System.Windows.Input.KeyEventHandler). Предварительное событие для события System.Windows.UIElement.KeyDown.
- **PreviewKeyUp** (System.Windows.Input.KeyEventHandler). Предварительное событие для события System.Windows.UIElement.KeyUp.

Также, как и для мыши, для клавиатуры существует статический класс, содержащий дополнительную функциональность в виде отдельного API — System.Windows.Input.Keyboard.

Статические свойства класса System.Windows.Input.Keyboard:

- **FocusedElement** (System.Windows.Input.IInputElement). Получает элемент, обладающий клавиатурным фокусом.
- **Modifiers** (System.Windows.Input.ModifierKeys). Получает набор нажатых в текущий момент клавиш управления.

Статические методы класса System.Windows.Input.Keyboard:

- **ClearFocus()**. Снимает клавиатурный фокус с элемента, обладающего ним.
- **Focus(element)**. Устанавливает клавиатурный фокус указанному элементу и возвращает элемент, обладающий клавиатурным фокусом.
- **GetKeyStates(key)**. Получает набор состояний для указанной клавиши.

- **IsKeyDown(key).** Проверяет, нажата ли указанная клавиша. Возвращает **true**, если указанная клавиша нажата; иначе — **false**.
- **IsKeyToggled(key).** Проверяет, включена ли указанная клавиша. Возвращает **true**, если указанная клавиша включена; иначе — **false**.
- **IsKeyUp(key).** Проверяет, отпущена ли указанная клавиша. Возвращает **true**, если указанная клавиша отпущена; иначе — **false**.

Для того, чтобы узнать какие клавиши управления нажаты необходимо использовать перечисление **System.Windows.Input.ModifierKeys**, которое содержит следующие варианты:

- Alt. Клавиша *Alt*.
- Control. Клавиша *Ctrl*.
- Shift. Клавиша *Shift*.
- Windows. Клавиша *Windows*.

Для того, чтобы узнать состояние клавиши на клавиатуре необходимо использовать перечисление **System.Windows.Input.KeyStates**, которое содержит следующие варианты:

- None. Клавиша отпущена.
- Pressed. Клавиша нажата.
- Toggled. Клавиша включена.

Для того, чтобы указать определенную виртуальную клавишу необходимо использовать перечисление **System.Windows.Input.Key**, которое содержит варианты для всех доступных клавиш.

### 14.2.1. Фокус

В WPF существует две концепции, связанные с фокусом: клавиатурный фокус и логический фокус. Клавиатурный фокус относится к элементу, который получает клавиатурный ввод, а логический относится к элементу в фокусной области видимости, который обладает фокусом. Элемент, обладающий клавиатурным фокусом, также обладает логическим фокусом, но элемент, обладающий логическим фокусом, не обязательно обладает клавиатурным фокусом.

Свойства класса `System.Windows.UIElement`, связанные с фокусом:

- **Focusable** (`System.Boolean`). Получает или задает значение, которое указывает, может ли элемент получать фокус. `True` если элемент может получать фокус; иначе — `false`. Значение по умолчанию зависит от конкретного производного класса.
- **IsFocused** (`System.Boolean`). Получает значение, которое указывает, обладает ли элемент логическим фокусом. `True` если элемент обладает логическим фокусом; иначе — `false`.
- **IsKeyboardFocused** (`System.Boolean`). Получает значение, которое указывает, обладает ли элемент клавиатурным фокусом. `True` если элемент обладает клавиатурным фокусом; иначе — `false`.
- **IsKeyboardFocusWithin** (`System.Boolean`). Получает значение, которое указывает, обладает ли элемент или один из дочерних элементов клавиатурным фокусом. `True` если элемент или один из дочерних элементов обладает клавиатурным фокусом; иначе — `false`.

События класса System.Windows.UIElement, связанные с фокусом:

- **FocusableChanged** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement.Focusable.
- **GotFocus** (System.Windows.RoutedEventHandler). Срабатывает, когда элемент получает логический фокус.
- **GotKeyboardFocus** (System.Windows.Input.KeyboardFocusChangedEventArgs). Срабатывает, когда элемент получает клавиатурный фокус.
- **IsKeyboardFocusedChanged** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement.IsKeyboardFocused.
- **IsKeyboardFocusWithinChanged** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement.IsKeyboardFocusWithin.
- **LostFocus** (System.Windows.RoutedEventHandler). Срабатывает, когда элемент теряет логический фокус.
- **LostKeyboardFocus** (System.Windows.Input.KeyboardFocusChangedEventArgs). Срабатывает, когда элемент теряет клавиатурный фокус.
- **PreviewGotKeyboardFocus** (System.Windows.Input.KeyboardFocusChangedEventArgs). Предварительное событие для события System.Windows.UIElement.GotKeyboardFocus.
- **PreviewLostKeyboardFocus** (System.Windows.Input.KeyboardFocusChangedEventArgs). Предваритель-

ное событие для события System.Windows.UIElement.LostKeyboardFocus.

Методы класса System.Windows.UIElement, связанные с фокусом:

- **Focus()**. Пытается установить фокус на элемент. Возвращает **true**, если указанный элемент получил и логический и клавиатурный фокус; **false**, если указанный элемент получил только логический фокус или попытка сменить фокус не удалась.

### Клавиатурный фокус

Как было сказано выше, если элемент обладает клавиатурным фокусом, то он получает клавиатурный ввод. Только один элемент в пределах всего рабочего стола может обладать клавиатурным фокусом. В WPF, для определения, обладает ли элемент таким фокусом, существует свойство System.Windows.UIElement.IsKeyboardFocused. Также получить элемент, обладающий клавиатурным фокусом в текущий момент, можно при помощи статического свойства System.Windows.Input.Keyboard.FocusedElement.

Для того, чтобы элемент обладал возможность получения клавиатурного фокуса, необходимо, чтобы его свойства System.Windows.UIElement.Focusable и System.Windows.UIElement.Visibile возвращали значение **true**.

Клавиатурный фокус может быть получен элементом посредством определенных манипуляций со стороны пользователя. Например, нажатие клавиши *Tab*, или нажатие кнопки мыши на определенном элементе. Программным образом можно переместить клавиатурный фокус на элемент при помощи статического метода System.Windows.Input.Keyboard.Focus. Этот метод пытается установить фокус

указанному элементу и возвращает элемент, обладающий клавиатурным фокусом, который может отличаться от указанного (например, новый или старый элемент заблокировали запрос на получение нового фокуса).

У базового класса элементов управления есть два свойства отвечающих за то, обладает ли элемент клавиатурным фокусом: System.Windows.UIElement.IsKeyboardFocused и System.Windows.UIElement.IsKeyboardFocusWithin. Первое указывает, обладает ли фокусом элемент, в то время как второе указывает, обладает ли фокусом элемент или один из его дочерних элементов.

Для того, чтобы установить начальный клавиатурный фокус при запуске приложения, необходимо, чтобы элемент, который должен обладать фокусом находился в визуальном дереве начального окна приложения и его свойства System.Windows.UIElement.Focusable и System.Windows.UIElement. IsVisible возвращали значение **true**. Рекомендуемым местом для установки начального фокуса является событие System.Windows.FrameworkElement.Loaded начального окна:

### C#

```
private void Window_Loaded(object sender,
                           RoutedEventArgs e)
{
    Keyboard.Focus(submitButton);
}
```

### Логический фокус

В пределах интерфейса приложения может быть несколько, так называемых, фокусных областей видимости.

Другими словами, можно разделить элементы на несколько областей, в пределах каждой из которых будет собственный элемент, обладающий фокусом. В данном случае это будет логический фокус. Когда клавиатурный фокус будет перемещаться из одной фокусной области видимости в другую, элемент, потерявший клавиатурный фокус, сохранит логический фокус. Когда клавиатурный фокус перемещается в другую фокусную область видимости он устанавливается элементу, обладающему логическим фокусом. Это позволяет реализовать переключение между несколькими группами логически разделенных элементов, с запоминанием элемента с фокусом в каждой группе, и восстановлением его при возврате.

В пределах приложения может быть несколько элементов, обладающих логическим фокусом, но только один, в пределах каждой фокусной области видимости. Элемент, обладающий клавиатурным фокусом, также обладает логическим фокусом в пределах той фокусной области видимости, к которой он принадлежит.

Для того, чтобы превратить элемент в фокусную область видимости, необходимо воспользоваться присоединенным свойством `System.Windows.Input.FocusManager`.`IsFocusScope`.

Пример создания фокусной области видимости из панели, используя разметку:

### XAML

```
<StackPanel FocusManager.IsFocusScope="True">
    <Button>Submit</Button>
    <Button>Cancel</Button>
</StackPanel>
```

Рассмотренной выше разметке соответствует следующий код:

C#

```
var submitButton = new Button { Content = "Submit" };
var cancelButton = new Button { Content = "Cancel" };

var stackPanel = new StackPanel();
stackPanel.Children.Add(submitButton);
stackPanel.Children.Add(cancelButton);

FocusManager.SetIsFocusScope(stackPanel, true);
```

Для того, чтобы получить фокусную область видимости необходимо воспользоваться статическим методом System.Windows.Input.**FocusManager**.GetFocusScope, указав элемент, для которого необходимо получить область видимости.

Для получения или задания фокусного элемента в пределах фокусной области видимости существуют соответствующие статические методы: System.Windows.Input.**FocusManager**.GetFocusElement и System.Windows.Input.**FocusManager**.SetFocusElement. Последний, обычно, используется для установки начального логического фокуса.

### 14.2.2. Клавиши доступа

Все элементы, представляющие из себя тот или иной вид кнопок (или пунктов меню) поддерживают работу с клавишами быстрого доступа, которые работают подобно мнемоническим командам элемента управления System.Windows.Controls.**Label**. Для этих целей используется точно такой же механизм с указанием символа подчеркивания перед необходимым символом.

# 15. Обзор базовых классов элементов визуального дерева

Как вы уже могли заметить, в WPF присутствует несколько базовых классов, содержащих много полезной функциональности, от которых наследуются все рассмотренные до этого момента панели и элементы управления. В данном разделе будет рассмотрена не описанная до этого функциональность некоторых из этих классов.

## 15.1. Класс **UIElement**

Свойства класса `System.Windows.UIElement`:

- **IsEnabled** (`System.Boolean`). Получает или задает значение, которое указывает, является ли элемент доступным для взаимодействия со стороны пользователя. `True` если элемент доступен для взаимодействия с пользователем; иначе — `false`. Значение по умолчанию — `true`.
- **IsVisible** (`System.Boolean`). Получает значение, которое указывает, отображается ли элемент. `True` если элемент видим; иначе — `false`.
- **Opacity** (`System.Double`). Получает или задает коэффициент прозрачности элемента. Данное свойство может принимать значения от 0.0 (прозрачный) до 1.0 (непрозрачный). Значение по умолчанию — 1.0.
- **SnapsToDevicePixels** (`System.Boolean`). Получает или задает значение, которое указывает, необходимо ли использо-

вать аппаратно-зависимые пиксели при отображении элемента. **True** если элемент должен отображаться в соответствии с аппаратно- зависимыми пикселями; иначе — **false**. Значение по умолчанию — **false**. При отображении элемента на мониторе с плотностью пикселей больше, чем 96 пикселей на дюйм, включение данного свойства позволит избежать нежелательных эффектов сглаживания, которое происходит, если рассчитанные размеры элемента не совпадают с аппаратными пикселями.

- **Visibility** (System.Windows.Visibility). Получает или задает режим отображения элемента. Значение по умолчанию — System.Windows.Visibility.Visible.  
События класса System.Windows.UIElement:
  - **IsEnabledChanged** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement.IsEnabled.
  - **IsVisibleChanged** (System.Windows.DependencyPropertyChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.UIElement. IsVisible.  
Методы класса System.Windows.UIElement:
    - **TranslatePoint**(point, relativeTo). Преобразует координаты указанные относительно текущего элемента в координаты относительно другого элемента.

Для того, чтобы указать режим отображения элемента необходимо использовать перечисление System.Windows. **Visibility**, которое содержит следующие варианты:

- Collapsed. Элемент не отображается и не резервирует за собой место в интерфейсе.

- **Hidden**. Элемент не отображается, но резервирует за собой место в интерфейсе.
- **Visible**. Элемент отображается.

## 15.2. Класс FrameworkElement

Свойства класса `System.Windows.FrameworkElement`:

- **ContextMenu** (`System.Windows.Controls.ContextMenu`). Получает или задает контекстное меню элемента.
- **IsInitialized** (`System.Boolean`). Получает значение, которое указывает, инициализирован ли объект. `True` если объект инициализирован; иначе — `false`.
- **IsLoaded** (`System.Boolean`). Получает значение, которое указывает, загружен ли элемент в визуальное дерево. `True` если элемент загружен в визуальное дерево; иначе — `false`.
- **Name** (`System.String`). Получает или задает имя элемента. Значение по умолчанию — `System.String.Empty`.
- **Parent** (`System.Windows.DependencyObject`). Получает родительский элемент.
- **Tag** (`System.Object`). Получает или задает произвольный объект, ассоциированный с текущим элементом, который может использоваться для хранения дополнительной информации, описывающей элемент.
- **ToolTip** (`System.Object`). Получает или задает подсказку элемента.
- **UseLayoutRounding** (`System.Boolean`). Получает или задает значение, которое указывает, необходимо ли округлять рассчитанные размеры и координаты для элемента на этапе компоновки. `True` если округле-

ние необходимо; иначе — **false**. Значение по умолчанию — **false**.

События класса System.Windows.FrameworkElement:

- **ContextMenuClosing** (System.Windows.Controls.ContextMenuEventHandler). Срабатывает непосредственно перед тем, как контекстное меню элемента закрывается.
- **ContextMenuOpening** (System.Windows.Controls.ContextMenuEventHandler). Срабатывает непосредственно перед тем, как контекстное меню элемента показывается.
- **Initialized** (System.EventHandler). Срабатывает, когда элемент инициализируется.
- **Loaded** (System.Windows.RoutedEventHandler). Срабатывает, когда элемент загружается в визуальное дерево, его размеры и позиция рассчитаны и он готов к отображению.
- **SizeChanged** (System.Windows.SizeChangedEventHandler). Срабатывает, когда изменяется значение свойства System.Windows.FrameworkElement.ActialHeight или System.Windows.FrameworkElement.ActialWidth.
- **ToolTipClosing** (System.Windows.Controls.ToolTipEventHandler). Срабатывает непосредственно перед тем, как подсказка элемента закрывается.
- **ToolTipOpening** (System.Windows.Controls.ToolTipEventHandler). Срабатывает непосредственно перед тем, как подсказка элемента открывается.
- **Unloaded** (System.Windows.RoutedEventHandler). Срабатывает, когда элемент выгружается из визуального дерева.

Методы класса System.Windows.**FrameworkElement**:

- **FindName(name)**. Получает элемент с указанным именем или **null**, если требуемый элемент не найден.

### 15.3. Класс Control

Свойства класса System.Windows.Controls.**Control**:

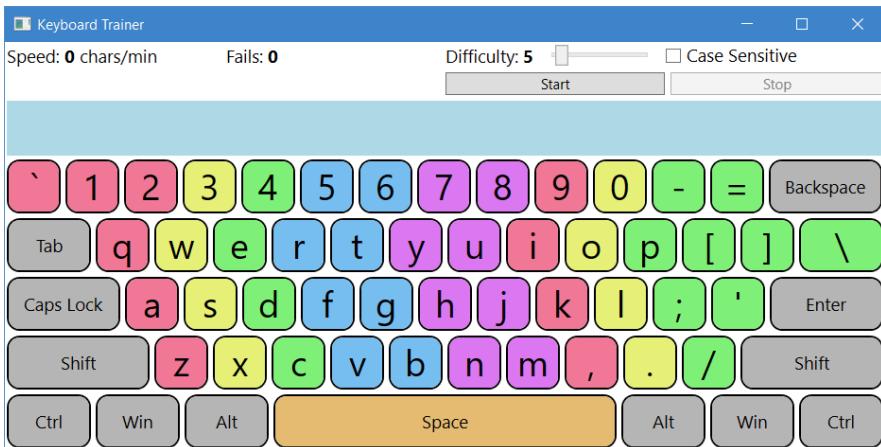
- **Background** (System.Windows.Media.Brush). Получает или задает кисть, которая используется для заливки заднего фона элемента. Значение по умолчанию — System.Windows.Media.Brushes.Transparent.
- **BorderBrush** (System.Windows.Media.Brush). Получает или задает кисть, которая используется для заливки рамки элемента. Значение по умолчанию — System.Windows.Media.Brushes.Transparent.
- **BorderThickness** (System.Windiws.Thickness). Получает или задает толщину рамки. Значение по умолчанию — объект System.Windows.Thickness, описывающий нулевую толщину со всех сторон.
- **FontFamily** (System.Windows.Media.FontFamily). Получает или задает семейство шрифтов.
- **FontSize** (System.Double). Получает или задает размер шрифта. Данное свойство может принимать только положительные значения. Значение по умолчанию — System.Windows.SystemFonts.MessageFontSize.
- **FontStretch** (System.Windows.FontStretch). Получает или задает режим растяжения шрифта. Значение по умолчанию — System.Windows.FontStretches.Normal.
- **FontStyle** (System.Windows.FontStyle). Получает или задает стиль шрифта. Значение по умолчанию — System.Windows.FontStyles.Normal.

- **FontWeight** (System.Windows.FontWeight). Получает или задает толщину шрифта. Значение по умолчанию — System.Windows.FontWeights.Normal.
- **Foreground** (System.Windows.Media.Brush). Получает или задает кисть, которая используется для заливки переднего фона элемента.
- **IsTabStop** (System.Boolean). Получает или задает значение, которое указывает, должен ли элемент учитываться при навигации клавишей *Tab*. *True* если элемент должен учитываться при навигации; иначе — *false*. Значение по умолчанию — *true*.
- **TabIndex** (System.Int32). Получает или задает порядок получения фокуса элементов при навигации клавишей *Tab*. Значение по умолчанию — System.Int32.MaxValue.

## 16. Домашнее задание

## Задание 1

Необходимо разработать приложение «Клавиатурный тренажер» (рис. 42). Главное окно приложения должно отображать клавиатуру с не интерактивными клавишами, которые необходимо для того, чтобы помогать пользователю ориентироваться на клавиатуре, не смотря на нее. При этом, при нажатии на каждую из клавиш, она должна подсвечиваться на экране. После запуска тренировочной сессии пользователю должна отобразиться произвольно сгенерированная строка для ввода, которая учитывает выбранный уровень сложности.



### Рис. 42. Задание 1

В верхней части окна должна отображаться статистическая информация: скорость набора корректного текста и количество допущенных ошибок. Также в верх-

ней части окна должны располагаться элементы управления, позволяющие настроить сложность генерируемой строки для ввода. При помощи «ползунка» пользователь может выбрать количество символов, которые должны использоваться для генерируемой строки. При этом можно указать необходимо ли генерировать строку с учетом регистра символов. В качестве используемых символов могут быть все символы, расположенные на рис. 43 и рис. 44.



**Рис. 43.** Задание 1

После нажатия на кнопку «Start» должна сгенерироваться строка символов с учетом заданных пользователем настроек. После этого нажимаемые пользователем клавиши должны учитываться и отображаться в виде введенных правильно символов либо в виде ошибок.

При нажатии на клавиши *Shift* и *Caps Lock*, клавиатура в приложении должна менять отображаемые символы, с учетом реально вводимых (рис. 44).

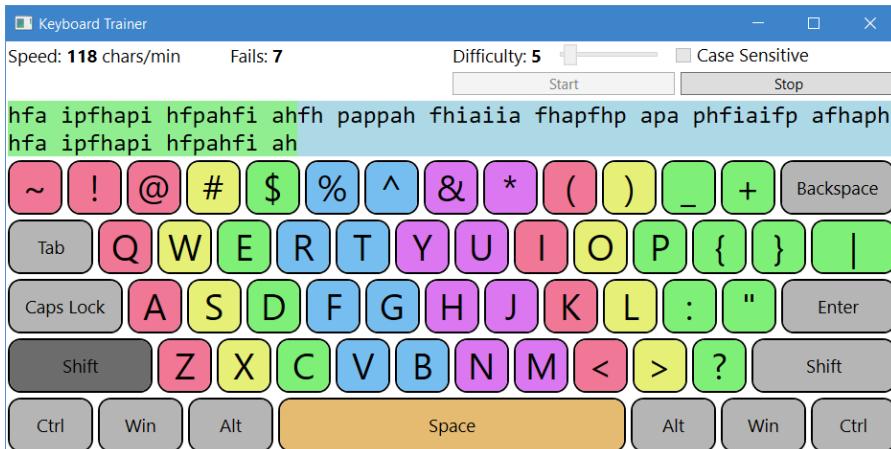


Рис. 44. Задание 1

Также необходимо предусмотреть выключение тех элементов управления, которые не могут использовать-ся в текущем состоянии приложения. Например, нельзя нажимать кнопку «Stop», если пользователь перед этим не нажал кнопку «Start».

Если в ходе выполнения задания возникнут труд-ности с созданием XAML-разметки, то можете восполь-зоваться пример разметки, который находится в папке **KeyboardTrainer** (архив с заданием прикреплен к PDF-файлу данного урока).





## Урок № 2

# Элементы управления

© Павел Дубский  
© Компьютерная Академия «Шаг».  
[www.itstep.org](http://www.itstep.org)

Все права на охраняемые авторским правом фото-, аудио- и видеопротивления, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.