

РФ



**Разработка приложений
с использованием WPF**

Урок № 3

Отображение документов
нефиксированного
формата.

Соединение данных
и элементов управления.

Управление стилями
и ресурсами

Содержание

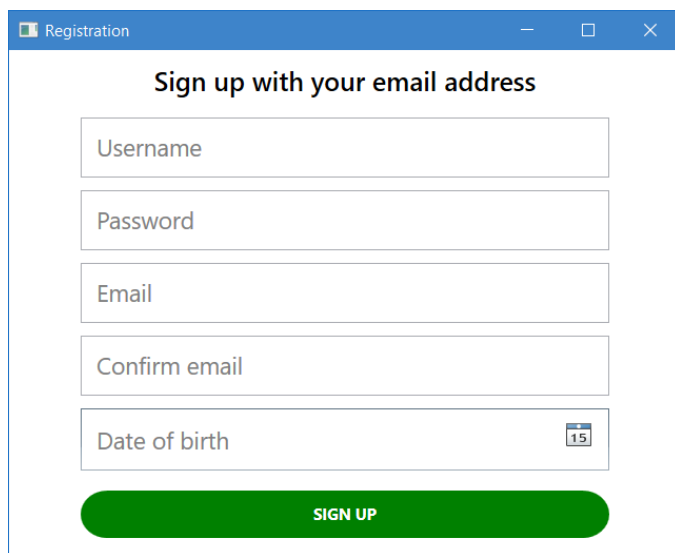
1. Элементы управления для работы с датами.....	4
1.1. Элемент управления Calendar.....	5
1.2. Элемент управления DatePicker.....	11
2. Панели инструментов.....	14
2.1. Элемент управления ToolBar.....	15
2.2. Элемент управления ToolBarTray.....	18
2.3. Переполнение.....	24
2.4. Позиционирование.....	27
2.5. Элементы управления панели инструментов ...	32
3. Элемент управления StatusBar.....	36
4. Обобщенный пример применения меню, панели инструментов и строки состояния.....	39
5. Текст и документы.....	47
5.1. Строчные текстовые элементы.....	47
5.2. Блочные текстовые элементы.....	54

5.2.1. Блочный текстовый элемент Paragraph.....	56
5.2.2. Блочный текстовый элемент List	58
5.2.3. Блочный текстовый элемент Table	65
5.2.4. Блочный текстовый элемент Section	73
5.3. Интеграция	74
6. Домашнее задание.....	78
6.1. Задание 1	78

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

1. Элементы управления для работы с датами

На сегодня, при создании практически любого приложения разработчики сталкиваются с необходимостью получения от пользователя информации в виде даты и/или времени. Существует множество ситуаций, когда это может понадобиться. Наиболее распространенной, вероятно, можно считать возможность пользователя регистрироваться в системе (приложение, веб-сайт, сервис и т.д.), создавая так называемую учетную запись (рис. 1).



The image shows a web browser window titled "Registration". Inside the window, the heading "Sign up with your email address" is centered. Below the heading are five input fields stacked vertically: "Username", "Password", "Email", "Confirm email", and "Date of birth". The "Date of birth" field has a small calendar icon to its right, displaying the number "15". At the bottom of the form is a large green button with the text "SIGN UP" in white capital letters.

Рис. 1. Пример окна регистрации нового пользователя

Одним из полей при такого рода регистрации, практически всегда, выступает поле «дата рождения». Создате-

лям приложения данная информация может понадобиться для того, чтобы более качественно предоставлять сервисы и услуги (например, выбор фильмов или сериалов, которые наиболее популярны среди определенной возрастной группы). Также эта информация может учитываться в лицензионном соглашении, которое также, обычно, указывает минимально допустимый возраст для пользователей.

В WPF присутствует ряд элементов управления для взаимодействия с датами (рис. 2). Они предназначены для отображения дат и календарей, а также для выбора их пользователем.

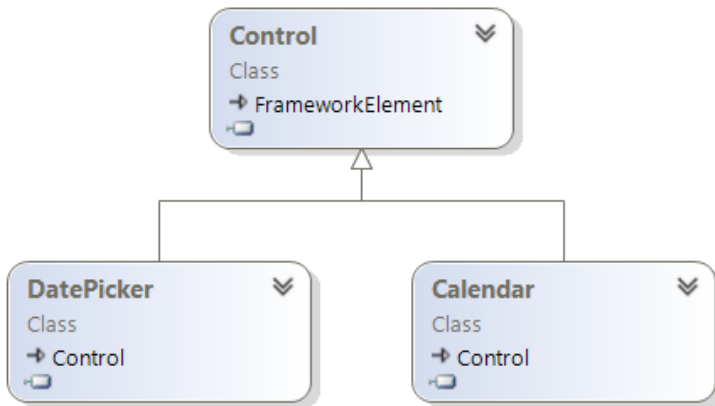


Рис. 2. Диаграмма классов, описывающих элементы управления для работы с датами

1.1. Элемент управления Calendar

Элемент управления `System.Windows.Controls.Calendar` позволяет отобразить календарь с возможностью выбора необходимой даты.

При использовании календаря, наиболее важным значением, как для пользователя, так и для программиста,

является выбранная пользователем дата. Для получения или задания ее программным образом элемент управления предоставляет свойство `System.Windows.Controls.Calendar.SelectedDate`.

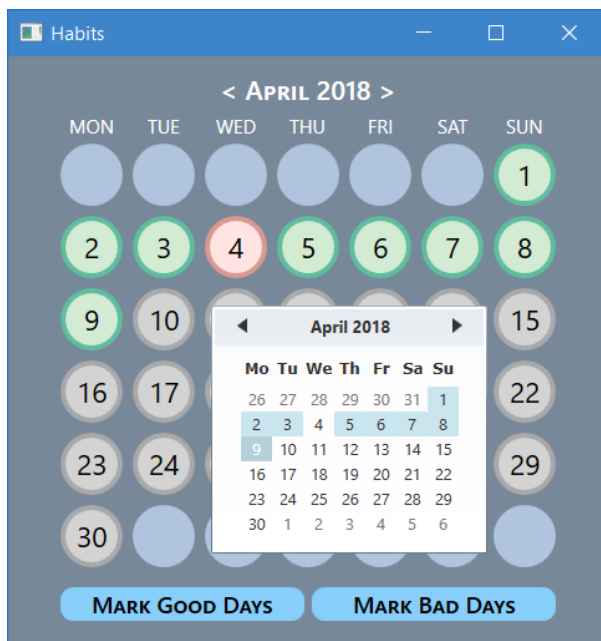


Рис. 3. Пример окна выбора нескольких дат

Большинству приложений хватает такой базовой функциональности полностью. Однако, все приложения разные, и некоторым требуются расширенные возможности при работе с календарем. В некоторых ситуациях пользователь должен иметь возможность выбора сразу нескольких дат, расположенных последовательно или вразброс. Например, разнообразные умные будильники активно используют эту особенность для того, чтобы пользователь обладал возможностью выставить будильник, который звонил

бы по определенным датам. Приложения, выполняющие роль ежедневников или календарей, также используют подобный режим выбора дат, позволяя пользователю, например, отметить в календаре дни своего отпуска или даты посещения врача.

Для того, чтобы задействовать описанную функциональность множественного выбора дат необходимо изменить режим выделения дат, который задается при помощи свойства `System.Windows.Controls.Calendar.SelectionMode`. Значение, установленное по умолчанию, позволяет выбирать только одну дату. Среди других вариантов есть возможность установки выбора дат, расположенных последовательно или вразброс. Также есть возможность отключения выбора полностью. В ситуациях, когда пользователь обладает возможностью выбора более, чем одной даты, необходимо использовать свойство `System.Windows.Controls.Calendar.SelectedDates` для их получения.

Для того, чтобы указать режим выделения дат необходимо использовать перечисление `System.Windows.Controls.CalendarSelectionMode`, которое содержит следующие варианты:

- **MultiRange**. Выделять можно несколько диапазонов дат. Для получения значения выделения необходимо использовать свойство `System.Windows.Controls.Calendar.SelectedDates`.
- **None**. Выделение запрещено.
- **SingleDate**. Выделять можно одну дату. Для получения значения выделения необходимо использовать свойство `System.Windows.Controls.Calendar.SelectedDate`.

- **SingleRange**. Выделять можно один диапазон дат. Для получения значения выделения необходимо использовать свойство `System.Windows.Controls.Calendar.SelectedDates`.

Другим не менее важным аспектом использования данного элемента управления является его возможность запретить выбор определенных дат. Это осуществляется путем добавления дат, которые должны быть помечены, как невыбираемые, в коллекцию, содержащуюся в свойстве `System.Windows.Controls.Calendar.BlackoutDates`. Это решение отлично подходит в ситуациях, когда недопустимые даты находятся вперемешку с допустимыми. Например, пользователь не должен иметь возможность выбирать даты, на которые попадают государственные праздники. Однако, часто встречается ситуация, что пользователь должен иметь возможность выбора даты только в определенном диапазоне, например, день своего рождения. В этой ситуации диапазон допустимых дат задается при помощи свойств `System.Windows.Controls.Calendar.DisplayDateStart` и `System.Windows.Controls.Calendar.DisplayDateEnd`, которые описывают начало и конец диапазона соответственно. При необходимости возможно указать только одну из границ диапазона, тем самым не ограничивая выбор в противоположном направлении.

Календарь предоставляет ряд свойств, которые позволят сделать использование элемента управления более удобным. Одним из таких свойств является свойство `System.Windows.Controls.Calendar.IsTodayHighlighted`, которое включает подсветку сегодняшней даты, что позволит пользователю лучше ориентироваться в календаре.

Несмотря на то, что согласно международному стандарту понедельник является первым днем недели, некоторые страны, среди которых США, Канада и Австралия, рассматривают воскресенье в качестве первого дня недели. Для изменения этой настройки можно воспользоваться свойством `System.Windows.Controls.Calendar.FirstDayOfWeek`.

Еще одной удобной функциональностью можно считать возможность задания даты, которую необходимо отображать при открытии календаря. Это позволяет открывать календарь на любой дате, которая наиболее всего уместна для ситуации пользователя. В качестве примера задействования данной особенности можно привести приложение, в котором пользователь должен выбрать дату для напоминания чего-либо. Приложение не было бы более «дружелюбным» если бы открывая календарь для выбора даты пользователю отображалась фиксированная дата, которая со временем будет все дальше и дальше от потенциально желаемой. За настройку данной функциональности отвечает свойство `System.Windows.Controls.Calendar.DisplayDate`. По умолчанию отображается сегодняшняя дата.

Элемент управления календарь обладает несколькими режимами отображения, что позволяет быстро и удобно осуществлять навигацию по датам. По умолчанию календарь отображает определенный месяц, но, при необходимости, это можно изменить при помощи свойства `System.Windows.Controls.Calendar.DisplayMode`.

Для того, чтобы указать режим отображения календаря необходимо использовать перечисление `System.Windows.`

Controls.[CalendarMode](#), которое содержит следующие варианты:

- **Decade.** Отображается десятилетие.
- **Month.** Отображается месяц.
- **Year.** Отображается год.

Приведенный ниже фрагмент разметки демонстрирует элемент управления (полный пример находится в папке [Wpf.Controls.Calendar.Xaml](#)):

XAML

```
<Grid VerticalAlignment="Center">
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <Calendar DisplayMode="Month" Grid.Column="0"/>
  <Calendar DisplayMode="Year" Grid.Column="1"/>
  <Calendar DisplayMode="Decade" Grid.Column="2"/>
</Grid>
```

Результат приведенной разметки показан на рис. 4.

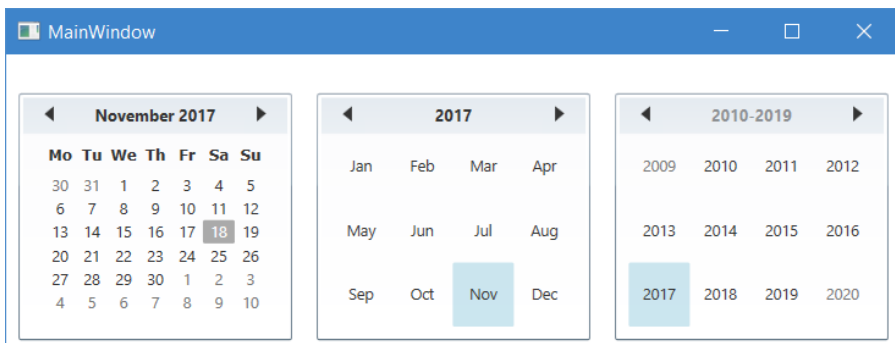


Рис. 4. Элемент управления Calendar

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

C#

```
var calendar1 = new Calendar { DisplayMode =  
                                CalendarMode.Month };  
var calendar2 = new Calendar { DisplayMode =  
                                CalendarMode.Year };  
var calendar3 = new Calendar { DisplayMode =  
                                CalendarMode.Decade };  
var grid = new Grid { VerticalAlignment =  
                       VerticalAlignment.Center };  
grid.ColumnDefinitions.Add(new ColumnDefinition());  
grid.ColumnDefinitions.Add(new ColumnDefinition());  
grid.ColumnDefinitions.Add(new ColumnDefinition());  
grid.Children.Add(calendar1);  
grid.Children.Add(calendar2);  
grid.Children.Add(calendar3);  
  
Grid.SetColumn(calendar1, 0);  
Grid.SetColumn(calendar2, 1);  
Grid.SetColumn(calendar3, 2);
```

1.2. Элемент управления DatePicker

Элемент управления `System.Windows.Controls.DatePicker` позволяет выбрать необходимую дату, пользуясь интерфейсом календаря или полем для ввода текста.

Данный элемент управления по большей части полагается на функциональность элемента управления календарь, который был рассмотрен ранее. Таким образом, многие его свойства просто перенаправляют вызов на соответствующие свойства календаря. К собственной

функциональности можно отнести ту, которая связана с ручным вводом выбранной даты. Так как ввод даты осуществляется посредством поля для ввода текста, то элемент управления обладает соответствующим свойством для хранения введенного текста — `System.Windows.Controls.DatePicker.Text`.

Также элемент управления позволяет выбрать формат отображения выбранной даты в поле для ввода текста, используя свойство `System.Windows.Controls.DatePicker.SelectedDateFormat`.

Для того, чтобы указать формат отображения выбранной даты необходимо использовать перечисление `System.Windows.Controls.DatePickerFormat`, которое содержит следующие варианты:

- **Long**. Указывает, что дни недели и месяца должны отображаться, используя несокращенные названия. Результат эквивалентен результату метода `System.DateTime.ToLongDateString`.
- **Short**. Указывает, что дни недели и месяца должны отображаться, используя сокращенные названия. Результат эквивалентен результату метода `System.DateTime.ToShortDateString`.

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

XAML

```
<DatePicker SelectedDate="{x:Static system:
    DateTime.Now}"/>
```

Результат приведенной выше разметки показан на рис. 5.

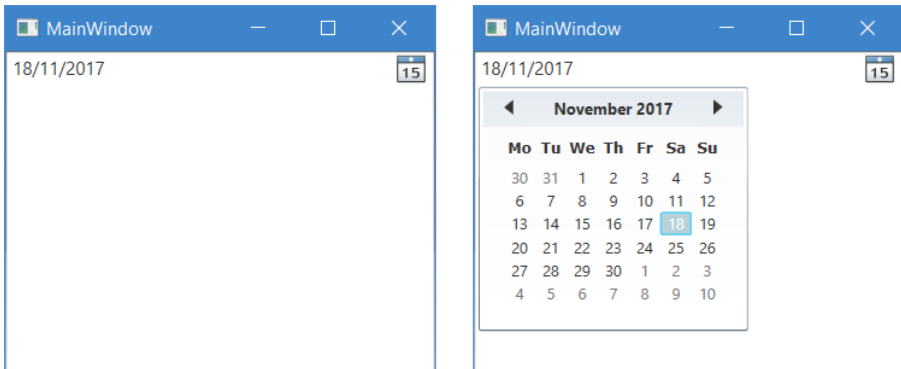


Рис. 5. Элемент управления DatePicker

В данном примере было использовано расширение разметки для того, чтобы указать в качестве значения атрибута значение статического свойства `System.DateTime.Now`: `SelectedDate="{x:Static system:DateTime.Now}"`.

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

C#

```
var datePicker = new DatePicker { SelectedDate =
                                DateTime.Now };
```

2. Панели инструментов

Панель инструментов представляет из себя ряд команд, чаще всего располагающихся под главным меню приложения. Обычно панель инструментов состоит из набора обычных кнопок, содержащих иконку и/или текст. WPF также предоставляет встроенную функциональность по изменению порядка панелей инструментов, и обработку переполнения их содержимого, описанную в нескольких стандартных классах (рис. 6).

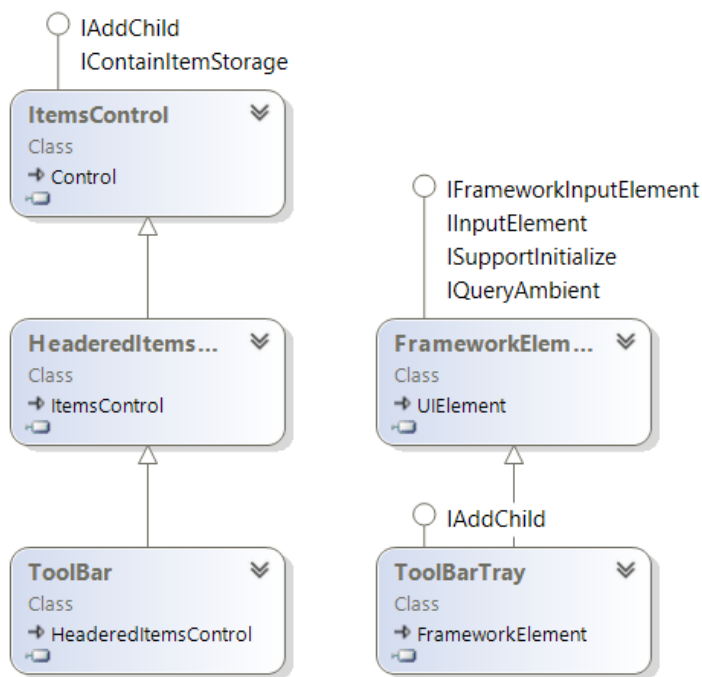


Рис. 6. Диаграмма классов, описывающих панели инструментов

2.1. Элемент управления **ToolBar**

Элемент управления `System.Windows.Controls.ToolTip` описывает панель инструментов.

Панель инструментов чаще всего представляется в виде горизонтальной полосы с расположенными на ней элементами управления. Однако, также встречается вариант вертикальных панелей инструментов, который может использоваться вместо горизонтальной или совместно с ней. Для того, чтобы изменить ориентацию панели инструментов необходимо использовать свойство `System.Windows.Controls.Orientation`, которое обладает значением по умолчанию, располагающим элемент управления горизонтально.

Панели инструментов предназначены для того, чтобы пользователь мог получить быстрый доступ к наиболее актуальным командам в текущем контексте выполнения приложения. Для реализации этой особенности важным является место расположения конкретных элементов внутри панели, а также место расположения самой панели в лотке (область, в которой хранятся панели инструментов). Позиционирование панели инструментов в лотке задается свойствами `System.Windows.Controls.ToolTip.Band` и `System.Windows.Controls.ToolTip.BandIndex`, которые указывают полосу в лотке и индекс в полосе лотка соответственно. Если эти значения не указывать явным образом, то панели инструментов будут располагаться друг за другом в одной полосе, в порядке их добавления.

При наличии большого количества элементов внутри панели инструментов или при изменении размеров ин-

терфейса приложения, может возникнуть ситуация, что не все элементы будут иметь возможность поместиться в область, предоставленную панели инструментов. В таком случае не поместившиеся элементы будут перенесены в специальную область, которая будет отображаться при необходимости или постоянно, в зависимости от значения, установленного для присоединенного свойства `System.Windows.Controls.ToolBar.OverflowMode`.

Для того, чтобы указать режим поведения элемента при переполнении необходимо использовать перечисление `System.Windows.Controls.OverflowMode`, которое содержит следующие варианты:

- **Always**. Элемент постоянно помещается в панель переполнения.
- **AsNeeded**. Элемент перемещается из панели инструментов в панель переполнения в зависимости от доступного пространства.
- **Never**. Элемент никогда не помещается в панель переполнения.

Для того, чтобы программным образом идентифицировать ситуацию с переполнением есть свойства `System.Windows.Controls.ToolBar.HasOverflowItems` и `System.Windows.Controls.ToolBar.IsOverflowOpen`, которые позволяют узнать, содержит ли панель инструментов элементы, которые не отображаются и отображается ли область с не поместившимися элементами соответственно.

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

XAML

```

<DockPanel LastChildFill="False">
    <ToolBar DockPanel.Dock="Top">
        <Button>Save</Button>
        <Button>Save As...</Button>
        <Button>Open...</Button>
        <Separator/>
        <Button>Cut</Button>
        <Button>Copy</Button>
        <Button>Paste</Button>
    </ToolBar>
</DockPanel>

```

Результат приведенной выше разметки показан на рис. 7.

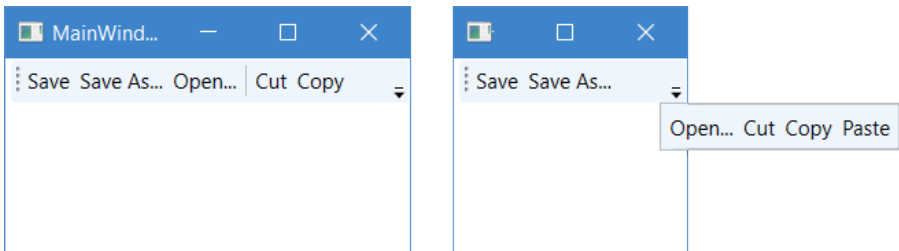


Рис. 7. Элемент управления ToolBar

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

C#

```

var button1 = new Button { Content = "Save" };
var button2 = new Button { Content = "Save As..." };
var button3 = new Button { Content = "Open..." };
var button4 = new Button { Content = "Cut" };

```

```

var button5 = new Button { Content = "Copy" };
var button6 = new Button { Content = "Paste" };
var separator = new Separator();
var toolBar = new ToolBar();
toolBar.Items.Add(button1);
toolBar.Items.Add(button2);
toolBar.Items.Add(button3);
toolBar.Items.Add(separator);
toolBar.Items.Add(button4);
toolBar.Items.Add(button5);
toolBar.Items.Add(button6);

var dockPanel = new DockPanel { LastChildFill = false };
dockPanel.Children.Add(toolBar);
DockPanel.SetDock(toolBar, Dock.Top);

```

2.2. Элемент управления ToolBarTray

Панель инструментов чаще всего помещается в «лоток», описанный элементом управления `System.Windows.Controls.ToolBarTray`. Данный элемент может содержать несколько разных панелей инструментов и предназначен для обработки их позиционирования. Свойство, отвечающее за позиционирование, выглядит и обрабатывается точно так же, как и у панели инструментов рассмотренной ранее.

Иногда возникает необходимость в закреплении всех или некоторых панелей инструментов на своем месте, без возможности перемещения их пользователем в пределах лотка. Для этого необходимо использовать свойство `System.Windows.Controls.ToolBarTray.IsLocked`, которое закрепляет на своих позициях все панели инструментов в лотке или присоединенное свойство `System.Windows.Controls.ToolBarTray.IsLocked`, которое может быть использовано

для одной или нескольких отдельных панелей, для закрепления только их. Если не использовать ни одно из этих свойств явным образом, то пользователь будет обладать возможностью перемещения панелей инструментов.

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

XAML

```
<DockPanel LastChildFill="False">
  <ToolBarTray DockPanel.Dock="Top">
    <ToolBar Band="1" BandIndex="1">
      <Button Height="30">
        <Image Source="Icons\Save.png"/>
      </Button>
      <Button Height="30">
        <Image Source="Icons\Print.png"/>
      </Button>
    </ToolBar>

    <ToolBar Band="1" BandIndex="2">
      <Button Height="30">
        <Image Source="Icons\Cut.png"/>
      </Button>
      <Button Height="30">
        <Image Source="Icons\Copy.png"/>
      </Button>
      <Button Height="30">
        <Image Source="Icons\Paste.png"/>
      </Button>
    </ToolBar>

    <ToolBar Band="2" BandIndex="1">
      <Button Height="30">
        <Image Source="Icons\Italic.png"/>
      </Button>
    </ToolBar>
  </ToolBarTray>
</DockPanel>
```

```

        <Button Height="30">
            <Image Source="Icons\Underline.png"/>
        </Button>
        <Button Height="30">
            <Image Source="Icons\Bold.png"/>
        </Button>
    </ToolBar>

    <ToolBar Band="2" BandIndex="2">
        <Button Height="30">
            <Image Source="Icons\AlignLeft.png"/>
        </Button>
        <Button Height="30">
            <Image Source="Icons\AlignCenter.png"/>
        </Button>
        <Button Height="30">
            <Image Source="Icons\AlignRight.png"/>
        </Button>
    </ToolBar>

</ToolBarTray>
</DockPanel>

```

Результат приведенной выше разметки показан на рис. 8.

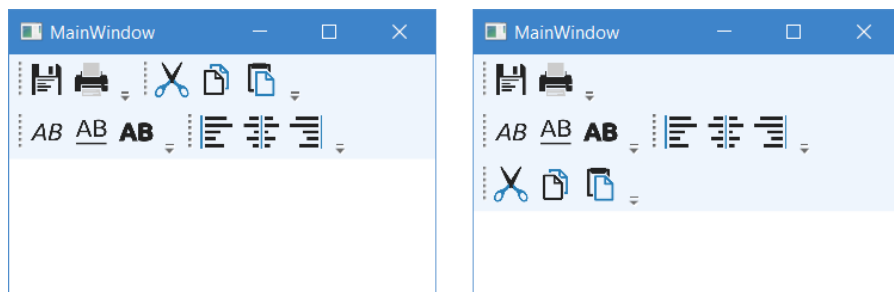


Рис. 8. Элемент управления ToolBarTray

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

C#

```
var image1 = new Image
{
    Source = new BitmapImage(new Uri(@"Icons\Save.png",
                                      UriKind.Relative))
};
var image2 = new Image
{
    Source = new BitmapImage(new Uri(@"Icons\Print.png",
                                      UriKind.Relative))
};
var image3 = new Image
{
    Source = new BitmapImage(new Uri(@"Icons\Cut.png",
                                      UriKind.Relative))
};
var image4 = new Image
{
    Source = new BitmapImage(new Uri(@"Icons\Copy.png",
                                      UriKind.Relative))
};
var image5 = new Image
{
    Source = new BitmapImage(new Uri(@"Icons\Paste.png",
                                      UriKind.Relative))
};
var image6 = new Image
{
    Source = new BitmapImage(new Uri(@"Icons\Italic.png",
                                      UriKind.Relative))
};
```

```

var image7 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Underline.png", UriKind.Relative))
};
var image8 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Bold.png", UriKind.Relative))
};
var image9 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\AlignLeft.png", UriKind.Relative))
};
var image10 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\AlignCenter.png", UriKind.Relative))
};
var image11 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\AlignRight.png", UriKind.Relative))
};

var button1 = new Button { Content = image1,
    Height = 30.0 };
var button2 = new Button { Content = image2,
    Height = 30.0 };
var button3 = new Button { Content = image3,
    Height = 30.0 };
var button4 = new Button { Content = image4,
    Height = 30.0 };
var button5 = new Button { Content = image5,
    Height = 30.0 };

```

```
var button6 = new Button { Content = image6,  
                           Height = 30.0 };  
var button7 = new Button { Content = image7,  
                           Height = 30.0 };  
var button8 = new Button { Content = image8,  
                           Height = 30.0 };  
var button9 = new Button { Content = image9,  
                           Height = 30.0 };  
var button10 = new Button { Content = image10,  
                           Height = 30.0 };  
var button11 = new Button { Content = image11,  
                           Height = 30.0 };  
  
var toolBar1 = new ToolBar { Band = 1, BandIndex = 1 };  
toolBar1.Items.Add(button1);  
toolBar1.Items.Add(button2);  
  
var toolBar2 = new ToolBar { Band = 1, BandIndex = 2 };  
toolBar2.Items.Add(button3);  
toolBar2.Items.Add(button4);  
toolBar2.Items.Add(button5);  
  
var toolBar3 = new ToolBar { Band = 2, BandIndex = 1 };  
toolBar3.Items.Add(button6);  
toolBar3.Items.Add(button7);  
toolBar3.Items.Add(button8);  
  
var toolBar4 = new ToolBar { Band = 2, BandIndex = 2 };  
toolBar4.Items.Add(button9);  
toolBar4.Items.Add(button10);  
toolBar4.Items.Add(button11);  
  
var toolBarTray = new ToolBarTray();  
toolBarTray.ToolBars.Add(toolBar1);  
toolBarTray.ToolBars.Add(toolBar2);  
toolBarTray.ToolBars.Add(toolBar3);  
toolBarTray.ToolBars.Add(toolBar4);
```

```
var dockPanel = new DockPanel { LastChildFill = false };  
dockPanel.Children.Add(toolBarTray);  
  
DockPanel.SetDock(toolBarTray, Dock.Top);
```

2.3. Переполнение

Одним из плюсов панели инструментов по сравнению с любой другой панелью, содержащей набор дочерних элементов, является встроенная функциональность обработки переполнения содержимым. Если содержимое панели инструментов не помещается в предоставленное ей пространство, то некоторые элементы будут автоматически помещены во всплывающем окне, которое будет доступно при нажатии на стрелку в нижнем правом углу панели инструментов.

При проектировании панели инструментов, некоторые ее элементы должны отображаться всегда, так как они предоставляют более важную функциональность. При этом другие могут быть существенно реже используемыми и, возможно, их захочется скрывать в выпадающем меню даже если есть свободное пространство. WPF позволяет сделать подобную настройку для каждого элемента панели инструментов, тем самым получив более дружелюбный интерфейс приложения.

Для управления поведением элемента в случае нехватки доступного пространства, чтобы отобразить все элементы панели инструментов, необходимо использовать присоединенное свойство `System.Windows.Controls.ToolBar.OverflowMode`. По умолчанию данное свойство содержит значение `System.Windows.Controls.OverflowMode.AsNeeded`,

которое означает, что элемент будет помещен в панель переполнения только в том случае, если панель не будет располагать доступным пространством для отображения всех элементов. Другими возможными значениями могут быть: `System.Windows.Controls.OverflowMode.Always` и `System.Windows.Controls.OverflowMode.Never`, которые означают, что элемент должен находиться в панели переполнения все время и никогда, соответственно.

Приведенный ниже фрагмент разметки демонстрирует переполнение панели инструментов (полный пример находится в папке `Wpf.ToolBars.Overflow.Xaml`):

XAML

```
<DockPanel LastChildFill="False">
  <ToolBarTray DockPanel.Dock="Top">
    <ToolBar>
      <Button>New</Button>
      <Button>Open</Button>
      <Button>Save</Button>
    </ToolBar>
    <ToolBar>
      <Button ToolBar.OverflowMode= "Always">
        Cut </Button>
      <Button ToolBar.OverflowMode= "AsNeeded">
        Copy </Button>
      <Button ToolBar.OverflowMode= "Never">
        Paste</Button>
    </ToolBar>
  </ToolBarTray>
</DockPanel>
```

Результат приведенной выше разметки показан на рис. 9.

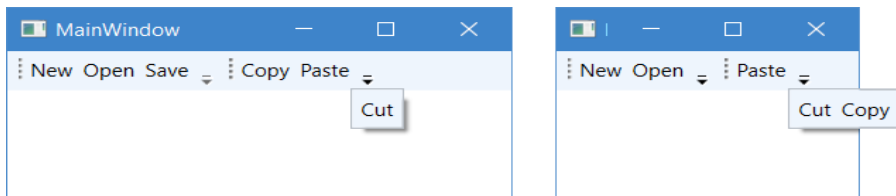


Рис. 9. Переполнение панели инструментов

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.ToolBars.Overflow.CSharp](#)):

C#

```
var button1 = new Button { Content = "New" };
var button2 = new Button { Content = "Open" };
var button3 = new Button { Content = "Save" };
var button4 = new Button { Content = "Cut" };
var button5 = new Button { Content = "Copy" };
var button6 = new Button { Content = "Paste" };

var toolBar1 = new ToolBar();
toolBar1.Items.Add(button1);
toolBar1.Items.Add(button2);
toolBar1.Items.Add(button3);

var toolBar2 = new ToolBar();
toolBar2.Items.Add(button4);
toolBar2.Items.Add(button5);
toolBar2.Items.Add(button6);

var toolBarTray = new ToolBarTray();
toolBarTray.ToolBars.Add(toolBar1);
toolBarTray.ToolBars.Add(toolBar2);
var dockPanel = new DockPanel { LastChildFill = false };
dockPanel.Children.Add(toolBarTray);

DockPanel.SetDock(toolBarTray, Dock.Top);
```

2.4. Позиционирование

В большинстве приложений панель инструментов позиционируется в верхней части окна. Чаще всего непосредственно под главным меню. При этом, панель инструментов может позиционироваться также внизу окна, или даже по его сторонам. Переместить панель в нижнюю часть окна довольно просто, для этого достаточно использовать панель `System.Windows.Controls.DockPanel` и свойство пристыковки с указанием необходимой границы. Однако, чтобы сделать панель инструментов вертикально ориентированной необходимо использовать соответствующее свойство — `System.Windows.Controls.ToolBarTray.Orientation`.

Приведенный ниже фрагмент разметки демонстрирует позиционирование панели инструментов (полный пример находится в папке `Wpf.ToolBars.Position.Xaml`):

XAML

```
<DockPanel LastChildFill="False">
  <ToolBarTray DockPanel.Dock="Top">
    <ToolBar>
      <Button Height="30">
        <Image Source="Icons\Save.png"/>
      </Button>
      <Button Height="30">
        <Image Source="Icons\Print.png"/>
      </Button>
    </ToolBar>

    <ToolBar>
      <Button Height="30">
        <Image Source="Icons\Italic.png"/>
      </Button>
    </ToolBar>
  </ToolBarTray>
</DockPanel>
```

```

        <Button Height="30">
            <Image Source="Icons\Underline.png"/>
        </Button>
        <Button Height="30">
            <Image Source="Icons\Bold.png"/>
        </Button>
    </ToolBar>
</ToolBarTray>

<ToolBarTray DockPanel.Dock="Left"
    Orientation="Vertical">
    <ToolBar>
        <Button Width="30">
            <Image Source="Icons\Cut.png"/>
        </Button>
        <Button Width="30">
            <Image Source="Icons\Copy.png"/>
        </Button>
        <Button Width="30">
            <Image Source="Icons\Paste.png"/>
        </Button>
    </ToolBar>
</ToolBarTray>

<ToolBarTray DockPanel.Dock="Right"
    Orientation="Vertical">
    <ToolBar>
        <Button Width="30">
            <Image Source="Icons\AlignLeft.png"/>
        </Button>
        <Button Width="30">
            <Image Source="Icons\AlignCenter.png"/>
        </Button>
        <Button Width="30">
            <Image Source="Icons\AlignRight.png"/>
        </Button>
    </ToolBar>
</ToolBarTray>
</DockPanel>

```

Результат приведенной выше разметки показан на рис. 10.

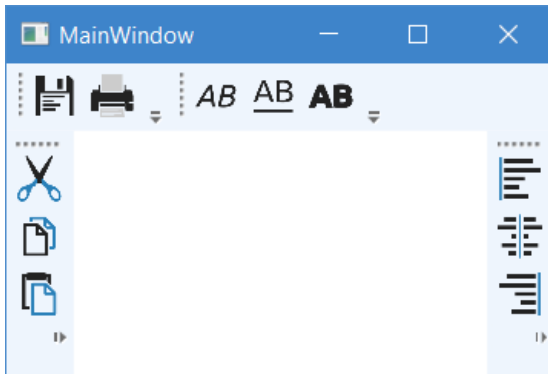


Рис. 10. Позиционирование панели инструментов

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.ToolBars.Position.CSharp](#)):

C#

```
var image1 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Save.png", UriKind.Relative))
};
var image2 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Print.png", UriKind.Relative))
};
var image3 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Italic.png", UriKind.Relative))
};
```

```
var image4 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Underline.png", UriKind.Relative))
};
var image5 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Bold.png", UriKind.Relative))
};
var image6 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Cut.png", UriKind.Relative))
};
var image7 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Copy.png", UriKind.Relative))
};
var image8 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Paste.png", UriKind.Relative))
};
var image9 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\AlignLeft.png", UriKind.Relative))
};
var image10 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\AlignCenter.png", UriKind.Relative))
};
var image11 = new Image
{
```

```

        Source = new BitmapImage(new Uri (
            @"Icons\AlignRight.png", UriKind.Relative))
    };

    var button1 = new Button { Content = image1,
                               Height = 30.0 };
    var button2 = new Button { Content = image2,
                               Height = 30.0 };
    var button3 = new Button { Content = image3,
                               Height = 30.0 };
    var button4 = new Button { Content = image4,
                               Height = 30.0 };
    var button5 = new Button { Content = image5,
                               Height = 30.0 };
    var button6 = new Button { Content = image6,
                               Width = 30.0 };
    var button7 = new Button { Content = image7,
                               Width = 30.0 };
    var button8 = new Button { Content = image8,
                               Width = 30.0 };
    var button9 = new Button { Content = image9,
                               Width = 30.0 };
    var button10 = new Button { Content = image10,
                                Width = 30.0 };
    var button11 = new Button { Content = image11,
                                Width = 30.0 };

    var toolBar1 = new ToolBar();
    toolBar1.Items.Add(button1);
    toolBar1.Items.Add(button2);

    var toolBar2 = new ToolBar();
    toolBar2.Items.Add(button3);
    toolBar2.Items.Add(button4);
    toolBar2.Items.Add(button5);

    var toolBar3 = new ToolBar();

```

```

toolBar3.Items.Add(button6);
toolBar3.Items.Add(button7);
toolBar3.Items.Add(button8);

var toolBar4 = new ToolBar();
toolBar4.Items.Add(button9);
toolBar4.Items.Add(button10);
toolBar4.Items.Add(button11);

var toolBarTray1 = new ToolBarTray();
toolBarTray1.ToolBars.Add(toolBar1);
toolBarTray1.ToolBars.Add(toolBar2);

var toolBarTray2 = new ToolBarTray { Orientation =
    Orientation.Vertical };
toolBarTray2.ToolBars.Add(toolBar3);

var toolBarTray3 = new ToolBarTray { Orientation =
    Orientation.Vertical };
toolBarTray3.ToolBars.Add(toolBar4);

var dockPanel = new DockPanel { LastChildFill = false };
dockPanel.Children.Add(toolBarTray1);
dockPanel.Children.Add(toolBarTray2);
dockPanel.Children.Add(toolBarTray3);

DockPanel.SetDock(toolBarTray1, Dock.Top);
DockPanel.SetDock(toolBarTray2, Dock.Left);
DockPanel.SetDock(toolBarTray3, Dock.Right);

```

2.5. Элементы управления панели инструментов

Панель инструментов может содержать любые элементы управления. Однако, это не означает, что стоит помещать в нее все подряд, без разбора. Некоторые элементы управления больше других уместны для использования в панели инструментов. К ним можно отнести

разнообразные кнопки, выпадающие списки и поля для ввода текста.

Приведенный ниже фрагмент разметки демонстрирует элементы управления панели инструментов (полный пример находится в папке [Wpf.ToolBars.CustomControls.Xaml](#)):

XAML

```
<DockPanel LastChildFill="False">
  <ToolBarTray DockPanel.Dock="Top">
    <ToolBar>

      <Button Height="22">
        <Image Source="Icons/Cut.png"/>
      </Button>
      <Button Height="22">
        <Image Source="Icons/Copy.png"/>
      </Button>
      <Button Height="22">
        <Image Source="Icons/Paste.png"/>
      </Button>
      <Separator/>

      <StackPanel Orientation="Horizontal">
        <TextBlock Text="Font family: "
          VerticalAlignment="Center"/>
        <ComboBox Width="80">
          <ComboBoxItem>Arial</ComboBoxItem>
          <ComboBoxItem>Calibri</ComboBoxItem>
          <ComboBoxItem>Consolas</ComboBoxItem>
        </ComboBox>
      </StackPanel>

    </ToolBar>
  </ToolBarTray>
</DockPanel>
```

Результат приведенной выше разметки показан на рис. 11.

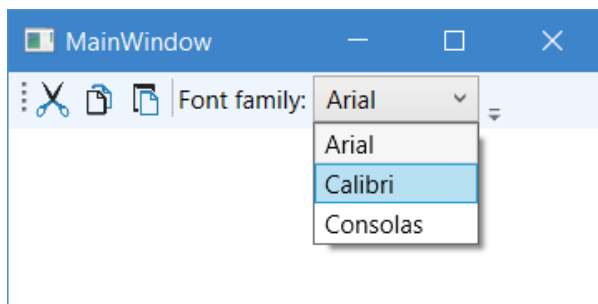


Рис. 11. Элементы управления панели инструментов

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.ToolBars.CustomControls.CSharp](#)):

C#

```
var image1 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Cut.png", UriKind.Relative))
};
var image2 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Copy.png", UriKind.Relative))
};
var image3 = new Image
{
    Source = new BitmapImage(new Uri(
        @"Icons\Paste.png", UriKind.Relative))
};
var button1 = new Button { Content = image1,
    Height = 22.0 };
```

```

var button2 = new Button { Content = image2,
                           Height = 22.0 };
var button3 = new Button { Content = image3,
                           Height = 22.0 };

var separator = new Separator();

var textBlock = new TextBlock
{
    Text = "Font family: ",
    VerticalAlignment = VerticalAlignment.Center
};

var comboBox = new ComboBox { Width = 80.0 };
comboBox.Items.Add("Arial");
comboBox.Items.Add("Calibri");
comboBox.Items.Add("Consolas");

var stackPanel = new StackPanel { Orientation =
    Orientation.Horizontal };
stackPanel.Children.Add(textBlock);
stackPanel.Children.Add(comboBox);

var toolBar = new ToolBar();
toolBar.Items.Add(button1);
toolBar.Items.Add(button2);
toolBar.Items.Add(button3);
toolBar.Items.Add(separator);
toolBar.Items.Add(stackPanel);

var toolBarTray = new ToolBarTray();
toolBarTray.ToolBars.Add(toolBar);

var dockPanel = new DockPanel { LastChildFill = false };
dockPanel.Children.Add(toolBarTray);

DockPanel.SetDock(toolBarTray, Dock.Top);

```

3. Элемент управления StatusBar

Элемент управления `System.Windows.Controls.Primitives.StatusBar` представляет из себя строку состояния, применяемую для отображения текущего состояния приложения или какого-либо из его процессов. Строка состояния чаще всего располагается в нижней части окна, в то время как главное меню и панели инструментов занимают верхнюю его часть.

Строка состояния относится к списковым элементам управления (рис. 12), что, в свою очередь, накладывает на нее те же самые требования по взаимодействию и структурированию разметки, что и на другие подобные элементы управления.

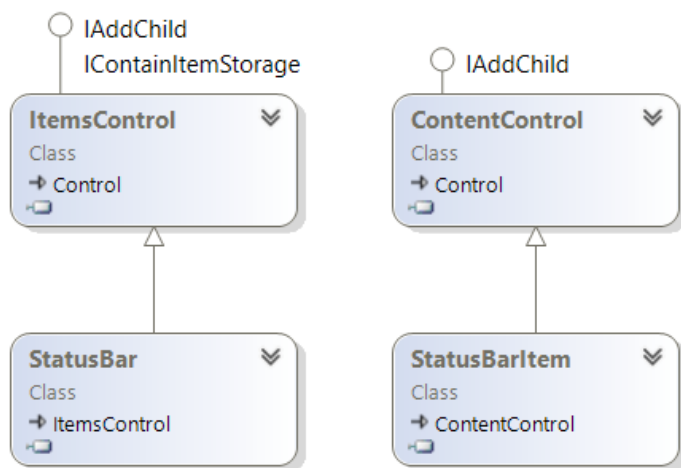


Рис. 12. Диаграмма классов, описывающих элементы управления для отображения строки состояния

В качестве контейнера элементов для элемента управления `System.Windows.Controls.Primitives.StatusBar` применяется класс `System.Windows.Controls.Primitives.StatusBarItem`.

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

XAML

```
<DockPanel LastChildFill="False">
  <StatusBar DockPanel.Dock="Bottom">
    <StatusBarItem>
      <TextBlock Text="Ln 1"/>
    </StatusBarItem>
    <Separator/>
    <StatusBarItem>
      <TextBlock Text="Col 8"/>
    </StatusBarItem>
    <Separator/>
    <StatusBarItem>
      <TextBlock Text="Ch 37"/>
    </StatusBarItem>
  </StatusBar>
</DockPanel>
```

Результат приведенной выше разметки показан на рис. 13.

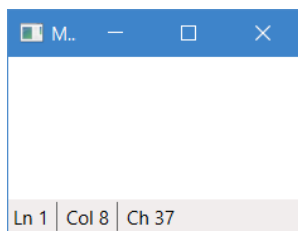


Рис. 13. Элемент управления StatusBar

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

C#

```
var texBlock1 = new TextBlock { Text = "Ln 1" };
var texBlock2 = new TextBlock { Text = "Col 8" };
var texBlock3 = new TextBlock { Text = "Ch 37" };

var statusBarItem1 = new StatusBarItem { Content =
    texBlock1 };
var statusBarItem2 = new StatusBarItem { Content =
    texBlock2 };
var statusBarItem3 = new StatusBarItem { Content =
    texBlock3 };

var separator1 = new Separator();
var separator2 = new Separator();

var statusBar = new StatusBar();
statusBar.Items.Add(statusBarItem1);
statusBar.Items.Add(separator1);
statusBar.Items.Add(statusBarItem2);
statusBar.Items.Add(separator2);
statusBar.Items.Add(statusBarItem3);

var dockPanel = new DockPanel { LastChildFill = false };
dockPanel.Children.Add(statusBar);

DockPanel.SetDock(statusBar, Dock.Bottom);
```

4. Обобщенный пример применения меню, панели инструментов и строки состояния

Для того, чтобы лучше понять принципы взаимодействия в пределах одного окна таких элементов управления, как меню и панель инструментов необходимо рассмотреть обобщенный пример. Так как панель инструментов чаще всего является способом быстрого доступа к наиболее часто используемым возможностям приложения, а меню позволяет добраться до всей функциональности приложения, их, обычно, используют вместе.

Строка состояния может использоваться для отображения подсказок, связанных с главным меню приложения, а также для информирования об общем состоянии приложения или отдельного его процесса.

Все упомянутые здесь элементы управления можно очень часто встретить в пределах одного и того же окна в приложениях, поэтому есть смысл рассмотреть пример их совместного использования.

Приведенный ниже фрагмент разметки демонстрирует обобщенный пример (полный пример находится в папке [Wpf.Example1.Xaml](#)):

XAML

```

<DockPanel>
  <Menu DockPanel.Dock="Top" IsMainMenu="True">
    <MenuItem Header="_File">
      <MenuItem Click="OnClickNew" Header="_New"/>
      <MenuItem Click="OnClickOpen"
        Header="_Open..." />
      <Separator/>
      <MenuItem Click="OnClickExit"
        Header="E_xit" />
    </MenuItem>

    <MenuItem Header="_Edit">
      <MenuItem Click="OnClickCut"
        Header="Cu_t" />
      <MenuItem Click="OnClickCopy"
        Header="_Copy" />
      <MenuItem Click="OnClickPaste"
        Header="_Paste" />
    </MenuItem>

    <MenuItem Header="_Format">
      <MenuItem Header="_Align">
        <MenuItem Click="OnClickAlignLeft"
          Header="_Left" />
        <MenuItem Click="OnClickAlignCenter"
          Header="_Center" />
        <MenuItem Click="OnClickAlignRight"
          Header="_Right" />
      </MenuItem>
    </MenuItem>
  </Menu>

  <ToolBarTray DockPanel.Dock="Top">
    <ToolBar>
      <Button Click="OnClickCut" Height="22">
        <Image Source="Icons/Cut.png" />
      </Button>
    </ToolBar>
  </ToolBarTray>
</DockPanel>

```



```

</Button>
<Button Click="OnClickCopy" Height="22">
    <Image Source="Icons/Copy.png"/>
</Button>
<Button Click="OnClickPaste" Height="22">
    <Image Source="Icons/Paste.png"/>
</Button>
</ToolBar>

<ToolBar>
    <Button Click="OnClickAlignLeft"
        Height="22">
        <Image Source="Icons/AlignLeft.png"/>
    </Button>

    <Button Click="OnClickAlignCenter"
        Height="22">
        <Image Source="Icons/AlignCenter.png"/>
    </Button>

    <Button Click="OnClickAlignRight"
        Height="22">
        <Image Source="Icons/AlignRight.png"/>
    </Button>
</ToolBar>
</ToolBarTray>

<StatusBar DockPanel.Dock="Bottom">
    <StatusBarItem>
        <TextBlock x:Name="currentLine" Text="Ln 1"
            Width="40"/>
    </StatusBarItem>

    <Separator/>

    <StatusBarItem>
        <TextBlock x:Name="currentColumn"
            Text="Col 1" Width="40"/>
    </StatusBarItem>
</StatusBar>

```

```

<TextBox x:Name="textBox"
        AcceptsReturn="True"
        HorizontalScrollBarVisibility="Auto"
        SelectionChanged="OnSelectionChanged"
        VerticalScrollBarVisibility="Auto"/>

</DockPanel>

```

Результат приведенной выше разметки показан на рис. 14.

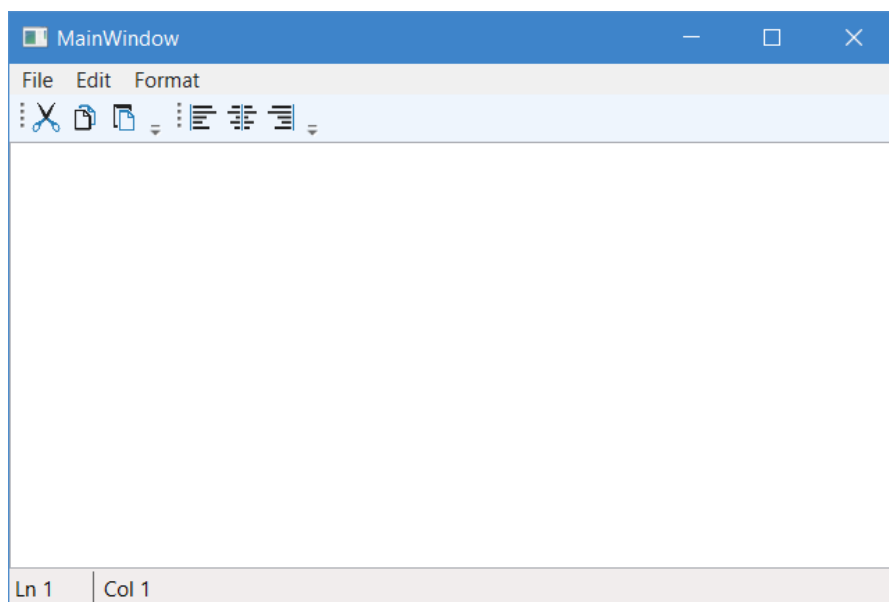


Рис. 14. Обобщенный пример применения меню, панели инструментов и строки состояния

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.Example1.CSharp](#)):

C#

```

var separator1 = new Separator();
var separator2 = new Separator();

var menuItem31 = new MenuItem { Header = "_Left" };
menuItem31.Click += OnClickAlignLeft;
var menuItem32 = new MenuItem { Header = "_Center" };
menuItem32.Click += OnClickAlignCenter;
var menuItem33 = new MenuItem { Header = "_Right" };
menuItem33.Click += OnClickAlignRight;

var menuItem21 = new MenuItem { Header = "_New" };
menuItem21.Click += OnClickOpen;
var menuItem22 = new MenuItem { Header = "_Open..." };
menuItem22.Click += OnClickOpen;
var menuItem23 = new MenuItem { Header = "E_xit" };
menuItem23.Click += OnClickExit;
var menuItem24 = new MenuItem { Header = "Cu_t" };
menuItem24.Click += OnClickCut;
var menuItem25 = new MenuItem { Header = "_Copy" };
menuItem25.Click += OnClickCopy;
var menuItem26 = new MenuItem { Header = "_Paste" };
menuItem26.Click += OnClickPaste;
var menuItem27 = new MenuItem { Header = "_Align" };
menuItem27.Items.Add(menuItem31);
menuItem27.Items.Add(menuItem32);
menuItem27.Items.Add(menuItem33);

var menuItem11 = new MenuItem { Header = "_File" };
menuItem11.Items.Add(menuItem21);
menuItem11.Items.Add(menuItem22);
menuItem11.Items.Add(separator1);
menuItem11.Items.Add(menuItem23);
var menuItem12 = new MenuItem { Header = "_Edit" };
menuItem12.Items.Add(menuItem24);
menuItem12.Items.Add(menuItem25);
menuItem12.Items.Add(menuItem26);

```

```

var menuItem13 = new MenuItem { Header = "_Format" };
menuItem13.Items.Add(menuItem27);
var menu = new Menu { IsMainMenu = true };
menu.Items.Add(menuItem11);
menu.Items.Add(menuItem12);
menu.Items.Add(menuItem13);
var image1 = new Image
{
    Source = new BitmapImage(new Uri("Icons/Cut.png",
                                     UriKind.Relative))
};
var image2 = new Image
{
    Source = new BitmapImage(new Uri("Icons/Copy.png",
                                     UriKind.Relative))
};
var image3 = new Image
{
    Source = new BitmapImage(new Uri("Icons/Paste.png",
                                     UriKind.Relative))
};
var image4 = new Image
{
    Source = new BitmapImage(new Uri("Icons/AlignLeft.png",
                                     UriKind.Relative))
};
var image5 = new Image
{
    Source = new BitmapImage(
        new Uri("Icons/AlignCenter.png",
                UriKind.Relative))
};
var image6 = new Image
{
    Source = new BitmapImage(
        new Uri("Icons/AlignRight.png",
                UriKind.Relative))
};

```

```

var button1 = new Button { Content = image1,
                           Height = 22.0 };
button1.Click += OnClickCut;
var button2 = new Button { Content = image2,
                           Height = 22.0 };
button2.Click += OnClickCopy;
var button3 = new Button { Content = image3,
                           Height = 22.0 };
button3.Click += OnClickPaste;
var button4 = new Button { Content = image4,
                           Height = 22.0 };
button4.Click += OnClickAlignLeft;
var button5 = new Button { Content = image5,
                           Height = 22.0 };
button5.Click += OnClickAlignCenter;
var button6 = new Button { Content = image6,
                           Height = 22.0 };
button6.Click += OnClickAlignRight;

var toolBar1 = new ToolBar();
toolBar1.Items.Add(button1);
toolBar1.Items.Add(button2);
toolBar1.Items.Add(button3);

var toolBar2 = new ToolBar();
toolBar2.Items.Add(button4);
toolBar2.Items.Add(button5);
toolBar2.Items.Add(button6);

var toolBarTray = new ToolBarTray();
toolBarTray.ToolBars.Add(toolBar1);
toolBarTray.ToolBars.Add(toolBar2);

currentLine = new TextBlock { Text = "Ln 1",
                              Width = 40.0 };
currentColumn = new TextBlock { Text = "Col 1",
                                Width = 40.0 };

```

```

var statusBarItem1 = new StatusBarItem { Content =
    currentLine };
var statusBarItem2 = new StatusBarItem { Content =
    currentColumnn };

var statusBar = new StatusBar();
statusBar.Items.Add(statusBarItem1);
statusBar.Items.Add(separator2);
statusBar.Items.Add(statusBarItem2);

textBox = new TextBox
{
    AcceptsReturn = true,
    HorizontalScrollBarVisibility =
        ScrollBarVisibility.Auto,
    VerticalScrollBarVisibility =
        ScrollBarVisibility.Auto
};
textBox.SelectionChanged += OnSelectionChanged;

var dockPanel = new DockPanel();
dockPanel.Children.Add(menu);
dockPanel.Children.Add(toolBarTray);
dockPanel.Children.Add(statusBar);
dockPanel.Children.Add(textBox);

DockPanel.SetDock(menu, Dock.Top);
DockPanel.SetDock(toolBarTray, Dock.Top);
DockPanel.SetDock(statusBar, Dock.Bottom);

```

5. Текст и документы

В данном разделе будет рассмотрена часть WPF, отвечающая за компоновку текста и наиболее распространенные виды форматирования. Также будут рассмотрены документы нефиксированного формата, которые используются для отображения большого объема текста, с автоматическим форматированием его для достижения наилучшей читабельности и использования доступного экранного пространства.

Текст, используемый совместно с элементами управления WPF, всегда является «обогащенным» (rich text). Это означает, что даже такой простой элемент, как `System.Windows.Controls.TextBlock` может содержать текст, отдельные фрагменты которого могут обладать разными настройками форматирования (шрифт, цвет, размер и т.д.).

5.1. Строчные текстовые элементы

Строчные текстовые элементы (рис. 15) предназначены для того, чтобы придать особый внешний вид определенному фрагменту строки, не затрагивая при этом остальной его части. Другими словами, такие элементы позволяют задать форматирование части строки.

Базовым классом для всех текстовых элементов является класс `System.Windows.Documents.TextElement`, который содержит все основные свойства форматирования, такие как: семейство шрифтов, размер текста, стиль и т.д. Эти же свойства можно обнаружить у большинства элементов управления, но на самом деле, обращения к ним просто

перенаправляют вызовы на свойства, объявленные в классе `System.Windows.Documents.TextElement`. Это сделано для того, чтобы избежать использования синтаксиса присоединенных свойств при работе с элементами управления.

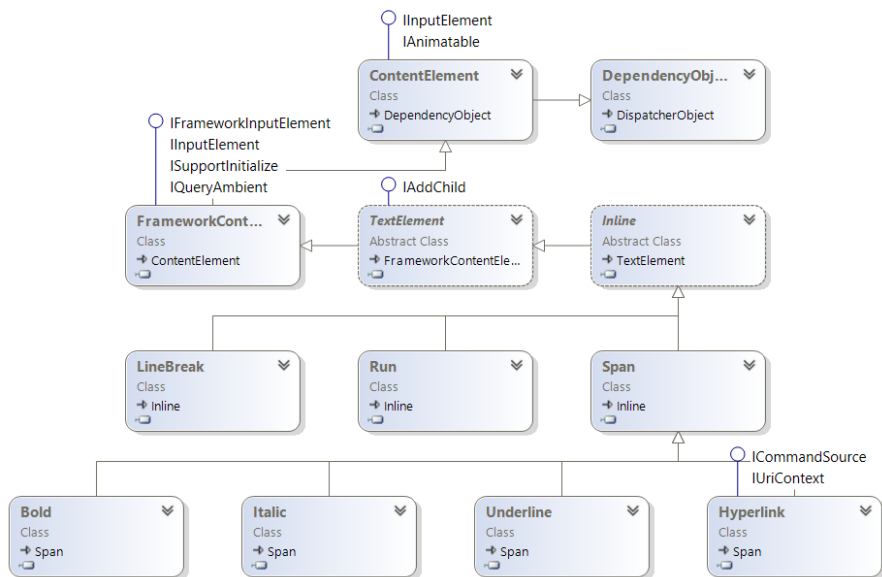


Рис. 15. Диаграмма классов, описывающих строчные текстовые элементы

Другим базовым классом для всех строчных текстовых элементов является класс `System.Windows.Documents.Inline`, который содержит функциональность, необходимую для описания элементов, используемых в пределах параграфа.

Большинство этих элементов позволяют всего лишь изменить форматирование для определенного набора символов строки, используя предустановленное поведение: жирный текст (`System.Windows.Documents.Bold`), курсивный текст (`System.Windows.Documents.Italic`), под-

черкнутый текст (`System.Windows.Documents.Underline`), текст-гиперссылка (`System.Windows.Documents.Hyperlink`).

Все эти текстовые элементы наследуются от базового класса `System.Windows.Documents.Span`, который не обладает никаким предустановленным видом форматирования, но при этом, может содержать коллекцию других строковых элементов, в качестве дочерних. Именно эта особенность и позволяет делать вложенность элементов друг в друга.

Есть также несколько строчных текстовых элементов, которые не наследуются от класса `System.Windows.Documents.Span`. К ним относятся `System.Windows.Documents.LineBreak` и `System.Windows.Documents.Run`.

Элемент `System.Windows.Documents.LineBreak` необходим для того, чтобы отобразить переход на новую строку в пределах одного параграфа.

Элемент `System.Windows.Documents.Run` необходим для того, чтобы содержать обычный текст. Этот элемент практически никогда не пишется явным образом. Однако, XAML-анализатор автоматически «оборачивает» весь текст именно в него, так как он единственный может содержать строку. Другие же строчные текстовые элементы, кроме разрыва строки, могут содержать только себе подобные элементы, но не обычный текст.

В качестве примера давайте рассмотрим следующий фрагмент XAML-разметки:

XAML

```
<TextBlock FontSize="25" Foreground="Blue">
  <Bold>W</Bold>indows <Bold>P</Bold>resentation
    <Bold>F</Bold>oundation
</TextBlock>
```

Рассмотренная выше разметка будет автоматически преобразована в следующую:

XAML

```
<TextBlock FontSize="25" Foreground="Blue">
    <Bold><Run Text="W"/></Bold><Run Text="indows "/>
    <Bold><Run Text="P"/></Bold><Run Text="resentation "/>
    <Bold><Run Text="F"/></Bold><Run Text="oundation"/>
</TextBlock>
```

Приведенный ниже фрагмент разметки демонстрирует строчные текстовые элементы (полный пример находится в папке `Wpf.Documents.Inlines.Xaml`):

XAML

```
<DockPanel>
    <Menu DockPanel.Dock="Top" IsMainMenu="True">
        <MenuItem Header="Inlines">
            <MenuItem>
                <MenuItem.Header>
                    <Italic>Italic</Italic>
                </MenuItem.Header>
            </MenuItem>

            <MenuItem>
                <MenuItem.Header>
                    <Bold>Bold</Bold>
                </MenuItem.Header>
            </MenuItem>

            <MenuItem>
                <MenuItem.Header>
                    <Underline>Underline</Underline>
                </MenuItem.Header>
            </MenuItem>
        </MenuItem>
    </Menu>
</DockPanel>
```

```

        <MenuItem>
            <MenuItem.Header>
                <Hyperlink>Hyperlink</Hyperlink>
            </MenuItem.Header>
        </MenuItem>
    </MenuItem>
</Menu>

<Button FontSize="35" Margin="5">
    <TextBlock TextWrapping="Wrap">
        Even a <Italic><Bold>simple</Bold></Italic>
        <Span FontFamily="Consolas"
            Foreground="Blue">Button
        </Span>
        can have some

        <Span>
            <Span.Foreground>
                <LinearGradientBrush EndPoint="1,1"
                    StartPoint="0,0">
                    <GradientStop Color="Red"
                        Offset="0"/>
                    <GradientStop Color="Green"
                        Offset="1"/>
                </LinearGradientBrush>
            </Span.Foreground>
            <Underline>advanced</Underline>
        </Span>
        text effects

    </TextBlock>
</Button>

</DockPanel>

```

Результат приведенной выше разметки показан на рис. 16.

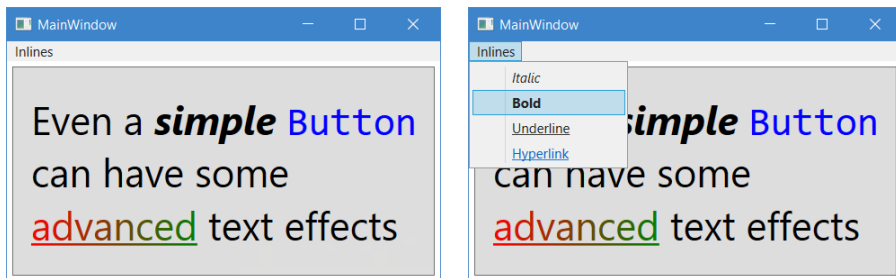


Рис. 16. Строчные текстовые элементы

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.Documents.Inlines.CSharp](#)):

C#

```
var italic = new Italic();
italic.Inlines.Add(new Run("Italic"));

var bold = new Bold();
bold.Inlines.Add(new Run("Bold"));

var underline = new Underline();
underline.Inlines.Add(new Run("Underline"));

var hyperlink = new Hyperlink();
hyperlink.Inlines.Add(new Run("Hyperlink"));

var menuItem21 = new MenuItem { Header = italic };
var menuItem22 = new MenuItem { Header = bold };
var menuItem23 = new MenuItem { Header = underline };
var menuItem24 = new MenuItem { Header = hyperlink };
var menuItem11 = new MenuItem { Header = "Inlines" };
menuItem11.Items.Add(menuItem21);
menuItem11.Items.Add(menuItem22);
menuItem11.Items.Add(menuItem23);
menuItem11.Items.Add(menuItem24);
```

```

var menu = new Menu { IsMainMenu = true };
menu.Items.Add(menuItem1);

var part1 = new Run("Even a ");
var part2 = new Italic(new Bold(new Run("simple")));
var part3 = new Run(" ");
var part4 = new Span(new Run("Button"))
{
    FontFamily = new FontFamily("Consolas"),
    Foreground = Brushes.Blue
};

var part5 = new Run(" can have some ");
var part6 = new Span(new Underline(new Run("advanced")))
{
    Foreground = new LinearGradientBrush(
        startColor: Colors.Red,
        endColor: Colors.Green,
        startPoint: new Point(0.0, 0.0),
        endPoint: new Point(1.0, 1.0)
    )
};

var part7 = new Run(" text effects");
var span = new Span();
span.Inlines.Add(part1);
span.Inlines.Add(part2);
span.Inlines.Add(part3);
span.Inlines.Add(part4);
span.Inlines.Add(part5);
span.Inlines.Add(part6);
span.Inlines.Add(part7);

var textBlock = new TextBlock(span) { TextWrapping =
    TextWrapping.Wrap };
var button = new Button
{

```

```

Content = textBlock,
FontSize = 35.0,
Margin = new Thickness(5.0)
};

var dockPanel = new DockPanel();
dockPanel.Children.Add(menu);
dockPanel.Children.Add(button);

DockPanel.SetDock(menu, Dock.Top);

```

5.2. Блочные текстовые элементы

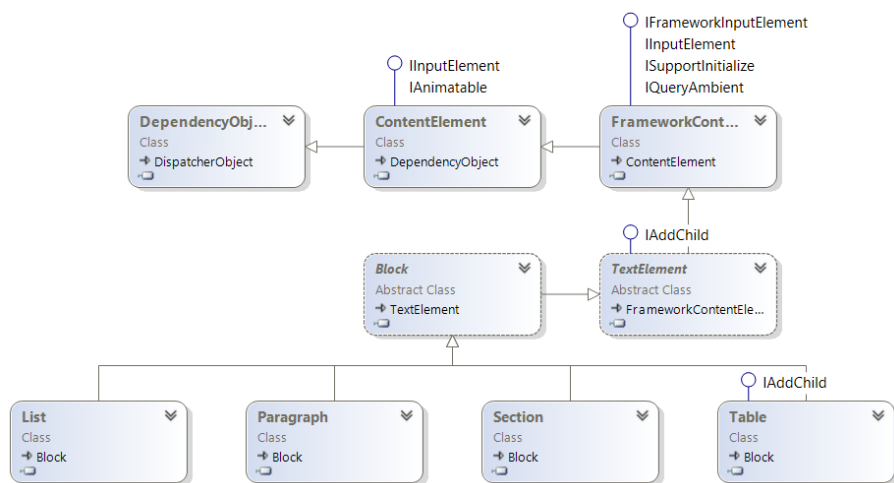


Рис. 17. Диаграмма классов,
описывающих блочные текстовые элементы

Не все текстовые элементы являются строчными. Также существуют блочные текстовые элементы (рис. 17), которые предназначены для описания структуры документа на более высоком уровне. К блочным

строчным элементам относятся таблицы, списки, а также параграфы.

Так как эти элементы не наследуются от класса `System.Windows.Documents.Inline`, они не могут быть использованы внутри элемента `System.Windows.Controls.TextBlock`. Для того, чтобы использовать блочные текстовые элементы необходимо воспользоваться одним из видов документов нефиксированного формата.

Все блочные элементы наследуются от класса `System.Windows.Documents.Block`.

Существует несколько элементов управления, которые способны отображать текст, используя блочные текстовые элементы.

- `System.Windows.Controls.FlowDocumentScrollViewer`. Этот элемент управления отображает текст в виде непрерывной вертикальной области с полосой прокрутки. Данный вид отображения очень похож на тот, что используется веб-браузерами для отображения HTML-страниц.
- `System.Windows.Controls.FlowDocumentPageViewer`. Этот элемент управления отображает текст, разбивая его на отдельные страницы и предоставляя элементы управления для навигации по ним. При отображении также происходит разбиение текста на колонки для повышения читабельности.
- `System.Windows.Controls.FlowDocumentReader`. Этот элемент управления предоставляет пользователю возможность выбрать один из двух вышеописанных стилей отображения.

5.2.1. Блочный текстовый элемент *Paragraph*

Блочный текстовый элемент `System.Windows.Documents.Paragraph` является простейшим из существующих и предназначен для хранения коллекции строчных текстовых элементов, среди которых может быть, как обычный текст, так и форматированный. Так как дочерние элементы являются строчными, обычный текст, помещенный в него, будет автоматически «обернут» в элемент `System.Windows.Documents.Run`.

Приведенный ниже фрагмент разметки демонстрирует блочный текстовый элемент (полный пример находится в папке `Wpf.Documents.Blocks.Paragraph.Xaml`):

XAML

```
<FlowDocument>

    <Paragraph FontSize="22">
        <Bold>Rules of Optimization</Bold>
    </Paragraph>

    <Paragraph>
        Rule 1: Don't do it
        <LineBreak/>
        Rule 2 (for experts only): Don't do it yet
    </Paragraph>

    <Paragraph TextAlignment="Right">
        <Italic>Michael A. Jackson</Italic>
    </Paragraph>

</FlowDocument>
```

Результат приведенной выше разметки показан на рис. 18.

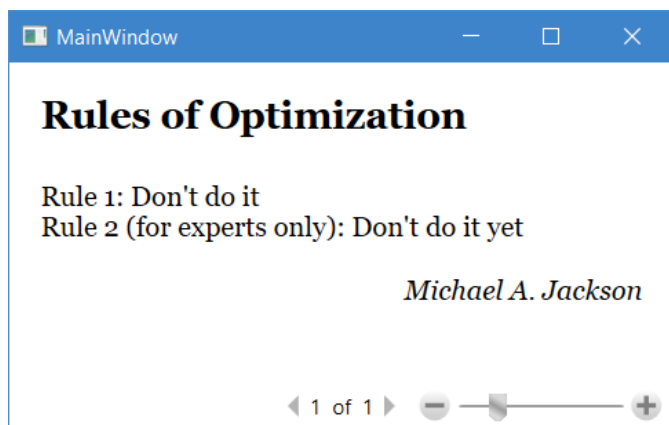


Рис. 18. Блочный текстовый элемент Paragraph

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.Documents.Blocks.Paragraph.CSharp](#)):

C#

```
var run1 = new Run("Rules of Optimization");
var run2 = new Run("Rule 1: Don't do it");
var run3 = new Run("Rule 2 (for experts only):
    Don't do it yet");
var run4 = new Run("Michael A. Jackson");

var paragraph1 = new Paragraph { FontSize = 22.0 };
paragraph1.Inlines.Add(new Bold(run1));

var paragraph2 = new Paragraph();
paragraph2.Inlines.Add(run2);
paragraph2.Inlines.Add(new LineBreak());
paragraph2.Inlines.Add(run3);

var paragraph3 = new Paragraph { TextAlignment =
    TextAlignment.Right };
paragraph3.Inlines.Add(run4);
```

```
var document = new FlowDocument();
document.Blocks.Add(paragraph1);
document.Blocks.Add(paragraph2);
document.Blocks.Add(paragraph3);
```

5.2.2. Блочный текстовый элемент *List*

Блочный текстовый элемент `System.Windows.Documents.List` позволяет отображать нумерованные и маркированные списки. WPF предоставляет возможность выбрать стиль отображения между буквами, цифрами (арабские или римские) и несколькими видами встроенных маркеров.

Приведенный ниже фрагмент разметки демонстрирует блочный текстовый элемент (полный пример находится в папке `Wpf.Documents.Blocks.List.Xaml`):

XAML

```
<FlowDocument>
  <Paragraph>
    <Bold>Checklist for good programming</Bold>
  </Paragraph>
  <List MarkerStyle="Disc">
    <ListItem>
      <Paragraph>
        Identifiers: Make sure all your
        identifiers are meaningful.
      </Paragraph>
      <List MarkerStyle="Decimal">
        <ListItem>
          <Paragraph>
            One-letter identifiers are
            almost never meaningful.
          </Paragraph>
        </ListItem>
      </List>
    </ListItem>
  </List>
</FlowDocument>
```

```

<ListItem>
  <Paragraph>
    Names like
    <Span FontFamily="Consolas"
      Foreground="Blue">flag
    </Span> and
    <Span FontFamily="Consolas"
      Foreground="Blue">temp
    </Span> are
      seldom meaningful. Instead of
    <Span FontFamily="Consolas"
      Foreground="Blue">flag
    </Span>, consider
      naming the Boolean condition
      it checks for, such as
    <Span FontFamily="Consolas"
      Foreground="Blue">valueFound
    </Span>.
  </Paragraph>
</ListItem>

<ListItem>
  <Paragraph>
    Consider multi-word identifiers, like
    <Span FontFamily="Consolas"
      Foreground="Blue">nameIndex
    </Span>.
    Long identifiers (within
    reason) tend to be very readable.
  </Paragraph>
</ListItem>
</List>
</ListItem>

<ListItem>
  <Paragraph>
    Bare literals: Avoid numbers

```

```

        other than
        <Span FontFamily="Consolas">0</Span> and
        <Span FontFamily="Consolas">1</Span>
        and strings other than
        <Span FontFamily="Consolas">""</Span>
        in your program except when you
        define constants.
    </Paragraph>
</List MarkerStyle="Decimal">

<ListItem>
    <Paragraph>
        Don't use a literal integer
        as an array bound.
    </Paragraph>
</ListItem>

<ListItem>
    <Paragraph>
        Don't use a literal integer to
        measure the size of a string or some
        data; use
        <Span FontFamily="Consolas"
            Foreground="Blue">sizeof()
        </Span> and
        <Span FontFamily="Consolas"
            Foreground="Blue">strlen()
        </Span> in C
        and C++ and
        <Span FontFamily="Consolas"
            Foreground="Blue">.Length
        </Span> in C#.
    </Paragraph>
</ListItem>

</List>
</ListItem>
</List>
</FlowDocument>

```

Результат приведенной выше разметки показан на рис. 19.

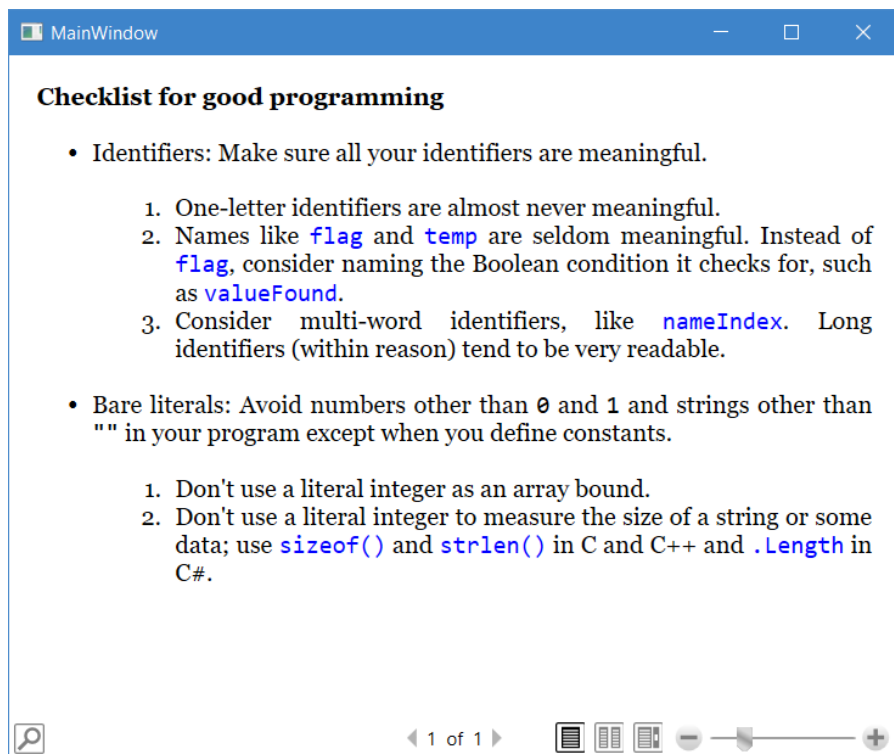


Рис. 19. Блочный текстовый элемент List

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.Documents.Blocks.List.CSharp](#)):

C#

```
var fontFamily = new FontFamily("Consolas");
var paragraph1 = new Paragraph();
paragraph1.Inlines.Add(new Bold(new Run(
    "Checklist for good programming")));
```

```

var paragraph2 = new Paragraph();
paragraph2.Inlines.Add(
    new Run("Identifiers: Make sure all your
            identifiers are meaningful.")
);

var paragraph3 = new Paragraph();
paragraph3.Inlines.Add(new Run("One-letter identifiers
                                are almost never meaningful.));
var paragraph4 = new Paragraph();
paragraph4.Inlines.Add(new Run("Names like "));
paragraph4.Inlines.Add(
    new Span(new Run("flag")) {
        FontFamily = fontFamily,
        Foreground = Brushes.Blue }
);
paragraph4.Inlines.Add(new Run(" and "));
paragraph4.Inlines.Add(
    new Span(new Run("temp")) {
        FontFamily = fontFamily,
        Foreground = Brushes.Blue }
);
paragraph4.Inlines.Add(new Run(" are seldom
                                meaningful. Instead of "));
paragraph4.Inlines.Add(
    new Span(new Run("flag")) {
        FontFamily = fontFamily,
        Foreground = Brushes.Blue }
);
paragraph4.Inlines.Add(
    new Run(", consider naming the Boolean condition
            it checks for, such as ")
);
paragraph4.Inlines.Add(
    new Span(new Run("valueFound")) {
        FontFamily = fontFamily,
        Foreground = Brushes.Blue }
);

```

```

paragraph4.Inlines.Add(new Run("."));

var paragraph5 = new Paragraph();
paragraph5.Inlines.Add(new Run("Consider multi-word
                                identifiers, like "));
paragraph5.Inlines.Add(
    new Span(new Run("nameIndex")) {
        FontFamily = fontFamily,
        Foreground = Brushes.Blue }
);
paragraph5.Inlines.Add(
    new Run(". Long identifiers (within reason)
            tend to be very readable.")
);

var paragraph6 = new Paragraph();
paragraph6.Inlines.Add(new Run("Bare literals: Avoid
                                numbers other than "));
paragraph6.Inlines.Add(new Span(new Run("0")) {
    FontFamily = fontFamily });
paragraph6.Inlines.Add(new Run(" and "));
paragraph6.Inlines.Add(new Span(new Run("1")) {
    FontFamily = fontFamily });
paragraph6.Inlines.Add(new Run(" and strings other
                                than "));
paragraph6.Inlines.Add(new Span(new Run("\\\\")) {
    FontFamily = fontFamily });
paragraph6.Inlines.Add(new Run(" in your program
                                except when you define constants."));

var paragraph7 = new Paragraph();
paragraph7.Inlines.Add(new Run("Don't use a literal
                                integer as an array bound."));

var paragraph8 = new Paragraph();
paragraph8.Inlines.Add(
    new Run(

```

```

        "Don't use a literal integer to measure
        the size of a string or some data; use "
    )
);
paragraph8.Inlines.Add(
    new Span(new Run("sizeof()")) {
        FontFamily = fontFamily,
        Foreground = Brushes.Blue }
);
paragraph8.Inlines.Add(new Run(" and "));
paragraph8.Inlines.Add(
    new Span(new Run("strlen()")) {
        FontFamily = fontFamily,
        Foreground = Brushes.Blue }
);
paragraph8.Inlines.Add(new Run(" in C and C++ and "));
paragraph8.Inlines.Add(
    new Span(new Run(".Length")) {
        FontFamily = fontFamily,
        Foreground = Brushes.Blue }
);
paragraph8.Inlines.Add(new Run(" in C#."));

var listItem1 = new ListItem(paragraph3);
var listItem2 = new ListItem(paragraph4);
var listItem3 = new ListItem(paragraph5);

var list1 = new List { MarkerStyle =
    TextMarkerStyle.Decimal };
list1.ListItems.Add(listItem1);
list1.ListItems.Add(listItem2);
list1.ListItems.Add(listItem3);

var listItem4 = new ListItem(paragraph7);
var listItem5 = new ListItem(paragraph8);

var list2 = new List { MarkerStyle =
    TextMarkerStyle.Decimal };

```



```
list2.ListItems.Add(listItem4);
list2.ListItems.Add(listItem5);

var listItem6 = new ListItem();
listItem6.Blocks.Add(paragraph2);
listItem6.Blocks.Add(list1);
var listItem7 = new ListItem();
listItem7.Blocks.Add(paragraph6);
listItem7.Blocks.Add(list2);

var list3 = new List { MarkerStyle =
    TextMarkerStyle.Disc };
list3.ListItems.Add(listItem6);
list3.ListItems.Add(listItem7);

var document = new FlowDocument();
document.Blocks.Add(paragraph1);
document.Blocks.Add(list3);
```

5.2.3. Блочный текстовый элемент *Table*

Блочный текстовый элемент `System.Windows.Documents.Table` позволяет форматировать отображение текста в виде таблицы. Частично функциональность этого элемента напоминает панель `System.Windows.Controls.Grid`, но панели не могут использоваться для форматирования структуры текста. При этом таблица обладает функциональностью, которой нет у сетки. Например, если таблица не помещается на одной странице, ее часть перенесется на одну или несколько следующих страниц. Также при описании таблицы не указывается изначально сколько именно должно быть строк и колонок, это количество вычисляется автоматически из описанной структуры. В качестве отрицательного эффекта использования та-

блицы является отсутствие возможности поместить несколько перекрывающихся элементов в одну и ту же ячейку. Каждая ячейка таблицы может содержать только один блочный строчный элемент. Это ограничение можно обойти, поместив в качестве такого элементе еще одну таблицу или список.

При описании таблицы необходимо соблюдать следующий принцип вложенности элементов. На самом верхнем уровне иерархии располагается элемент `System.Windows.Documents.Table`, описывающий, собственно, всю таблицу целиком. Данный элемент содержит коллекцию дочерних элементов, каждый из которых описывает набор строк, логически сгруппированных вместе, и представленный в виде класса `System.Windows.Documents.TableRowGroup`. Каждый вложенный элемент (строка), описывается элементом `System.Windows.Documents.TableRow`, и содержит коллекцию ячеек таблицы. Каждая ячейка описывается отдельным экземпляром класса `System.Windows.Documents.TableCell` и может содержать набор блочных текстовых элементов, которые определяют непосредственное содержимое таблицы, отображаемое на экране.

Приведенный ниже фрагмент разметки демонстрирует блочный текстовый элемент (полный пример находится в папке `Wpf.Documents.Blocks.Table.Xaml`):

XAML

```
<FlowDocument>
  <Table TextAlignment="Center">
    <TableRowGroup Background="Wheat"
      FontWeight="Bold">
```

```

<TableRow>
  <TableCell ColumnSpan="5"
    FontSize="20">
    <Paragraph>Logical Operators
    </Paragraph>
  </TableCell>
</TableRow>

<TableRow>
  <TableCell>
    <Paragraph>A</Paragraph>
  </TableCell>
  <TableCell>
    <Paragraph>B</Paragraph>
  </TableCell>
  <TableCell>
    <Paragraph>A & & B
    </Paragraph>
  </TableCell>
  <TableCell>
    <Paragraph>A || B</Paragraph>
  </TableCell>
  <TableCell>
    <Paragraph>!A</Paragraph>
  </TableCell>
</TableRow>
</TableRowGroup>

<TableRowGroup Background="AliceBlue"
  FontFamily="Consolas"
  Foreground="Blue">
  <TableRow>
    <TableCell>
      <Paragraph>>false</Paragraph>
    </TableCell>
    <TableCell>
      <Paragraph>>false</Paragraph>
    </TableCell>
  </TableRow>
</TableRowGroup>

```

```

        </TableCell>
        <TableCell>
            <Paragraph>false</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>false</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
    </TableRow>

    <TableRow>
        <TableCell>
            <Paragraph>false</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>false</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
    </TableRow>

    <TableRow>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>false</Paragraph>
        </TableCell>
    </TableRow>

```

```

        <TableCell>
            <Paragraph>false</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>false</Paragraph>
        </TableCell>
    </TableRow>

    <TableRow>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>true</Paragraph>
        </TableCell>
        <TableCell>
            <Paragraph>false</Paragraph>
        </TableCell>
    </TableRow>

</TableRowGroup>
</Table>
</FlowDocument>

```

Результат приведенной выше разметки показан на рис. 20.

Logical Operators				
A	B	A && B	A B	!A
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Рис. 20. Блочный текстовый элемент Table

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.Documents.Blocks.Table.CSharp](#)):

C#

```
var cell111 = new TableCell(new Paragraph(new
Run("Logical Operators")))
{
    ColumnSpan = 5,
    FontSize = 20.0
};
var cell121 = new TableCell(new Paragraph(
    new Run("A")));
var cell122 = new TableCell(new Paragraph(
    new Run("B")));
var cell123 = new TableCell(new Paragraph(
    new Run("A && B")));
var cell124 = new TableCell(new Paragraph(
    new Run("A || B")));
var cell125 = new TableCell(new Paragraph(
    new Run("!A")));
var cell131 = new TableCell(new Paragraph(
    new Run("false")));
var cell132 = new TableCell(new Paragraph(
    new Run("false")));
```

```
var cell133 = new TableCell(new Paragraph(  
    new Run("false")));  
var cell134 = new TableCell(new Paragraph(  
    new Run("false")));  
var cell135 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell141 = new TableCell(new Paragraph(  
    new Run("false")));  
var cell142 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell143 = new TableCell(new Paragraph(  
    new Run("false")));  
var cell144 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell145 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell151 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell152 = new TableCell(new Paragraph(  
    new Run("false")));  
var cell153 = new TableCell(new Paragraph(  
    new Run("false")));  
var cell154 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell155 = new TableCell(new Paragraph(  
    new Run("false")));  
var cell161 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell162 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell163 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell164 = new TableCell(new Paragraph(  
    new Run("true")));  
var cell165 = new TableCell(new Paragraph(  
    new Run("true")));  
  
var row1 = new TableRow();
```

```
row1.Cells.Add(cell111);

var row2 = new TableRow();
row2.Cells.Add(cell121);
row2.Cells.Add(cell122);
row2.Cells.Add(cell123);
row2.Cells.Add(cell124);
row2.Cells.Add(cell125);

var row3 = new TableRow();
row3.Cells.Add(cell131);
row3.Cells.Add(cell132);
row3.Cells.Add(cell133);
row3.Cells.Add(cell134);
row3.Cells.Add(cell135);

var row4 = new TableRow();
row4.Cells.Add(cell141);
row4.Cells.Add(cell142);
row4.Cells.Add(cell143);
row4.Cells.Add(cell144);
row4.Cells.Add(cell145);

var row5 = new TableRow();
row5.Cells.Add(cell151);
row5.Cells.Add(cell152);
row5.Cells.Add(cell153);
row5.Cells.Add(cell154);
row5.Cells.Add(cell155);

var row6 = new TableRow();
row6.Cells.Add(cell161);
row6.Cells.Add(cell162);
row6.Cells.Add(cell163);
row6.Cells.Add(cell164);
row6.Cells.Add(cell165);
```



```

var rowGroup1 = new TableRowGroup
{
    Background = Brushes.Wheat,
    FontWeight = FontWeights.Bold
};
rowGroup1.Rows.Add(row1);
rowGroup1.Rows.Add(row2);

var rowGroup2 = new TableRowGroup
{
    Background = Brushes.AliceBlue,
    FontFamily = new FontFamily("Consolas"),
    Foreground = Brushes.Blue
};
rowGroup2.Rows.Add(row3);
rowGroup2.Rows.Add(row4);
rowGroup2.Rows.Add(row5);
rowGroup2.Rows.Add(row6);

var table = new Table { TextAlignment =
    TextAlignment.Center };
table.RowGroups.Add(rowGroup1);
table.RowGroups.Add(rowGroup2);

var document = new FlowDocument();
document.Blocks.Add(table);

```

5.2.4. Блочный текстовый элемент Section

Блочный текстовый элемент `System.Windows.Documents.Section` предназначен для объединения нескольких блочных текстовых элементов вместе. Это может понадобиться для упрощения настроек форматирования группы элементов. Также этот элемент может использоваться для логической группировки.

5.3. Интеграция

WPF разделяет все свои элементы на две группы: элементы наследуемые от `System.Windows.FrameworkElement` и элементы наследуемые от `System.Windows.FrameworkContentElement`. К первым относятся элементы управления, такие как кнопки, поля для ввода текста и др., а к последним элементы для описания содержимого, такие как параграфы, таблицы и строчные текстовые элементы. Такое разделение позволяет отдельным группам элементов использовать различные модели компоновки.

В прошлых примерах было рассмотрено каким образом можно поместить элементы содержимого внутри элементов управления. Обратный вариант интеграции также возможен. Для этого необходимо воспользоваться одним из двух элементов, которые способны содержать элементы управления:

- `System.Windows.Documents.BlockUIContainer`. Блочный элемент.
- `System.Windows.Documents.InlineUIContainer`. Строчный элемент.

Приведенный ниже фрагмент разметки демонстрирует интеграцию элементов управления в документ нефиксированного формата (полный пример находится в папке `Wpf.Documents.Integration.Xaml`):

XAML

```
<FlowDocument>
  <Paragraph>
    We build our computer (systems) the way we
    build our cities: over time, without a
```

```

        plan, on top of ruins.
    </Paragraph>

    <Paragraph TextAlignment="Right">
        <Italic>Ellen Ullman</Italic>
    </Paragraph>

    <BlockUIContainer Margin="0,10,0,0">
        <StackPanel HorizontalAlignment="Right"
            Orientation="Horizontal">
            <Button>Like</Button>
            <Button Margin="10,0,0,0">Don't Like
        </StackPanel>
    </BlockUIContainer>
</FlowDocument>

```

Результат приведенной выше разметки показан на рис. 21.

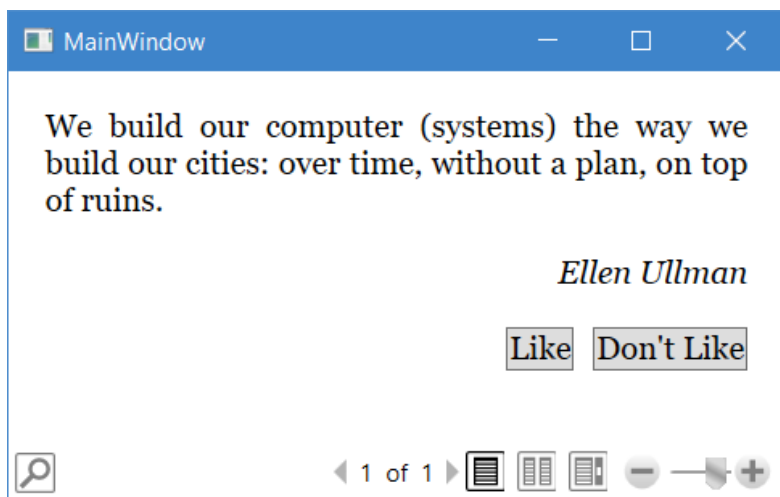


Рис. 21. Интеграция элементов управления в документ нефиксированного формата

Рассмотренной выше разметке соответствует следующий код (полный пример находится в [Wpf.Documents.Integration.CSharp](#)):

C#

```
var button1 = new Button { Content = "Like" };
var button2 = new Button
{
    Content = "Don't Like",
    Margin = new Thickness(10.0, 0.0, 0.0, 0.0)
};

var stackPanel = new StackPanel
{
    HorizontalAlignment = HorizontalAlignment.Right,
    Orientation = Orientation.Horizontal
};

stackPanel.Children.Add(button1);
stackPanel.Children.Add(button2);

var uiContainer = new BlockUIContainer
{
    Child = stackPanel,
    Margin = new Thickness(0.0, 10.0, 0.0, 0.0)
};

var paragraph1 = new Paragraph(
    new Run(
        "We build our computer (systems) the way we
        build our cities: over time, without a plan,
        on top of ruins."
    )
);

var paragraph2 = new Paragraph(new Italic(
    new Run("Ellen Ullman")));
```

```
var document = new FlowDocument();  
document.Blocks.Add(paragraph1);  
document.Blocks.Add(paragraph2);  
document.Blocks.Add(uiContainer);
```

6. Домашнее задание

6.1. Задание 1

Необходимо разработать приложения, которое позволяло бы устанавливать разнообразные настройки форматирования заданным участкам текста (рис. 22).

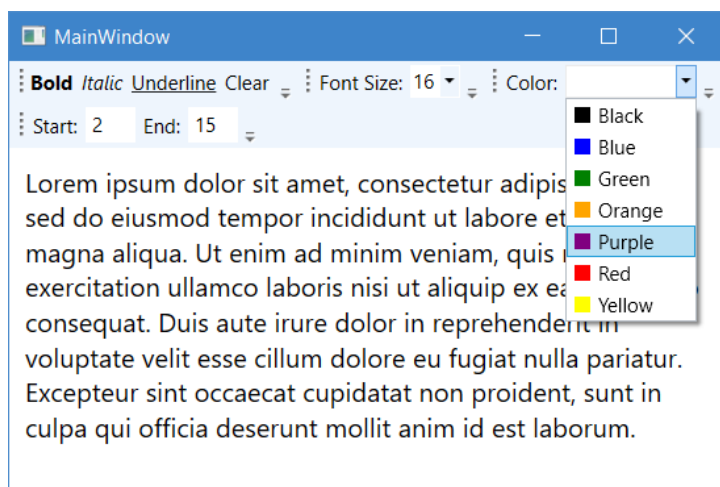


Рис. 22. Задание 1

В верхней части окна должна располагаться панель инструментов, содержащая элементы управления форматирования текста.

Для указания диапазона (фрагмента) текста необходимо использовать два поля для ввода текста, в которых будут задаваться индексы первого и последнего символа входящих в диапазон. Данные поля для ввода текста должны позволять вводить только положительные числа и ноль.

В случае указания некорректного диапазона (например, начальный индекс больше конечного или индекс за пределами текста), все остальные элементы управления на панели задач должны стать неактивными.

- Кнопка «**Bold**» должна делать текст в выбранном диапазоне жирным.
- Кнопка «*Italic*» должна делать текст в выбранном диапазоне курсивным.
- Кнопка «Underline» должна делать текст в выбранном диапазоне подчеркнутым.
- Кнопка «**Clear**» должна снимать все установленные настройки форматирования для текста в выбранном диапазоне (стиль текст, цвет и размер шрифта).

При выборе элемента из выпадающих списков, отвечающих за размер шрифта и его цвет, необходимо устанавливать соответствующее форматирование тексту в выбранном диапазоне.

Настройки форматирования могут накладываться друг на друга. Например, если в выбранном диапазоне содержится подчеркнутый текст и пользователь нажимает кнопку «*Italic*», то текст останется подчеркнутым, но при этом станет еще и курсивным.

Если в ходе выполнения задания возникнут трудности с созданием XAML-разметки, то можете воспользоваться пример разметки, который находится в папке **DocumentEditor**.



Урок № 3

Отображение документов нефиксированного формата. Соединение данных и элементов управления. Управление стилями и ресурсами

© Павел Дубский

© Компьютерная Академия «Шаг».

www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объёме, 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». Объём 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.