1. Дополнительные возможности меню, настройка шрифта, выравнивания и цвета: TEXTEDIT, версия 2

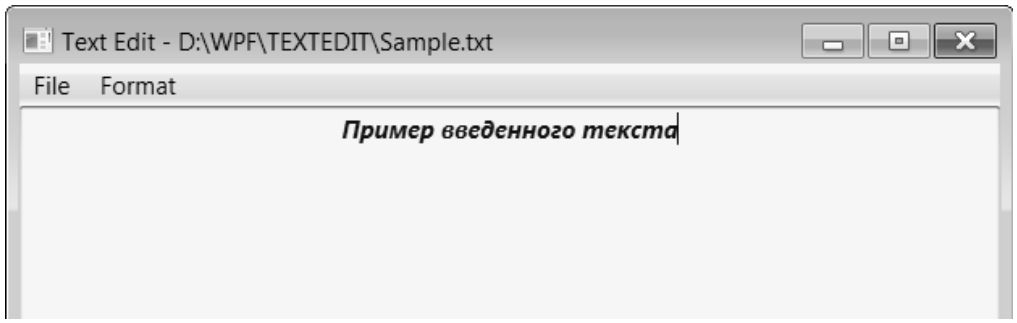

Рис. 1. Верхняя часть окна приложения TEXTEDIT версии 2

1.1. Установка начертания символов. Команды меню – флажки

```
<Window x:Class="TEXTEDIT.MainWindow"
    … PreviewKeyDown="Window_PreviewKeyDown" >
  …
  <DockPanel >
    <Menu DockPanel.Dock="Top" >
      <MenuItem x:Name="file1" Header="_File" >
        …
      </MenuItem>
      <MenuItem x:Name="format1" Header="_Format" >
        <MenuItem x:Name="bold1" Header="_Bold"
            InputGestureText="Ctrl+B" Click="bold1_Click" />
        <MenuItem x:Name="italic1" Header="_Italic"
            InputGestureText="Ctrl+I" Click="italic1_Click" />
        <MenuItem x:Name="underline1" Header="_Underline"
            InputGestureText="Ctrl+U" Click="underline1_Click" />
      </MenuItem>
    </Menu>
    …
  </DockPanel>
</Window>
```

Определите в классе MainWindow новые обработчики, указанные в xaml-файле:

```
private void bold1_Click(object sender, RoutedEventArgs e)
{
    bold1.IsChecked = !bold1.IsChecked;
    textBox1.FontWeight = bold1.IsChecked ? FontWeights.Bold :
      FontWeights.Normal;
}
private void italic1_Click(object sender, RoutedEventArgs e)
{
    italic1.IsChecked = !italic1.IsChecked;
    textBox1.FontStyle = italic1.IsChecked ? FontStyles.Italic :
      FontStyles.Normal;
}
private void underline1_Click(object sender, RoutedEventArgs e)
{
    underline1.IsChecked = !underline1.IsChecked;
    textBox1.TextDecorations = underline1.IsChecked ?
      TextDecorations.Underline : null;
}

private void Window_PreviewKeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyboardDevice.Modifiers == ModifierKeys.Control)
        switch (e.Key)
        {
            case Key.B:
                bold1_Click(null, null);
                e.Handled = true;
                break;
            case Key.I:
                italic1_Click(null, null);
                e.Handled = true;
                break;
            case Key.U:
                underline1_Click(null, null);
                e.Handled = true;
                break;
        }
}
```

Результат. При выполнении добавленных в меню команд устанавливается соответствующий *стиль* (начертание) символов в редакторе: «Bold» – **полужирный**, «Italic» – *курсив*, «Underline» – подчеркнутый. При

вызове меню «Format» около команд с установленными стилями изображаются «галочки» (как в случае флажков-переключателей). При повторном выполнении команды соответствующий стиль отменяется и «галочка» исчезает. Поскольку в нашем редакторе форматные настройки не сохраняются вместе с текстом, выполнение команд форматирования не влияет на значение свойства Modified.

1.2. Установка выравнивания текста. Команды меню – радиокнопки

```
<Window x:Class="TEXTEDIT.MainWindow"
    … >
  …
  <DockPanel >
    <Menu DockPanel.Dock="Top" >
      <MenuItem x:Name="file1" Header="_File" >
        …
      </MenuItem>
      <MenuItem x:Name="format1" Header="_Format" >
        <MenuItem x:Name="bold1" Header="_Bold"
            InputGestureText="Ctrl+B" Click="bold1_Click" />
        <MenuItem x:Name="italic1" Header="_Italic"
            InputGestureText="Ctrl+I" Click="italic1_Click" />
        <MenuItem x:Name="underline1" Header="_Underline"
            InputGestureText="Ctrl+U" Click="underline1_Click" />
        <Separator/>
        <MenuItem x:Name="leftJustify1" Header="_Left justify"
            InputGestureText="Ctrl+L"
            Click="leftJustify1_Click" />
        <MenuItem x:Name="center1" Header="_Center"
            InputGestureText="Ctrl+E"
            Click="leftJustify1_Click" />
        <MenuItem x:Name="rightJustify1" Header="_Right justify"
            InputGestureText="Ctrl+R"
            Click="leftJustify1_Click" />
      </MenuItem>
    </Menu>
    …
  </DockPanel>
</Window>
```

В класс MainWindow добавьте новое поле:

```
MenuItem alignItem;
```

В конструктор класса MainWindow добавьте новые операторы:

```
alignItem = leftJustify1;
leftJustify1.IsChecked = true;
leftJustify1.Tag = HorizontalAlignment.Left;
center1.Tag = HorizontalAlignment.Center;
rightJustify1.Tag = HorizontalAlignment.Right;
```

Определите в классе MainWindow новый обработчик leftJustify1_Click (этот обработчик указан во всех трех добавленных пунктах меню):

```
private void leftJustify1_Click(object sender, RoutedEventArgs e)
{
    MenuItem mi = sender as MenuItem;
    if (mi.IsChecked)
        return;
    alignItem.IsChecked = false;
    alignItem = mi;
    mi.IsChecked = true;
    textBox1.HorizontalContentAlignment =
      (HorizontalAlignment)mi.Tag;
}
```

В оператор switch обработчика Window_PreviewKeyDown добавьте новые варианты клавиатурных комбинаций:

```
case Key.L:
    leftJustify1_Click(leftJustify1, null);
    e.Handled = true;
    break;
case Key.E:
    leftJustify1_Click(center1, null);
    e.Handled = true;
    break;
case Key.R:
    leftJustify1_Click(rightJustify1, null);
    e.Handled = true;
    break;
```

Результат. При выполнении добавленных в меню команд устанавливается соответствующее *выравнивание* текста в редакторе: «Left justify» – по левому краю, «Center» – по центру, «Right justify» – по правому краю. При вызове меню Format около команды с текущим выравниванием изображается «галочка».

1.3. Установка цвета символов и фона. Определение новых команд WPF и использование диалогового окна из библиотеки Windows Forms

Для действий по выбору цвета шрифта и цвета фона отсутствуют стандартные команды WPF, однако их несложно реализовать самостоятельно. Так как действия по настройке пунктов меню «традиционным» способом (основанным на обработчиках событий) мы уже описали ранее в этой версии программы TEXTEDIT, для новых пунктов меню выберем второй подход, связав их с командами WPF, которые предварительно определим.

Для определения новых команд надо создать новый класс. Оформим его в виде отдельного файла.

Выполните команду Project | Add Class…, в появившемся диалоговом окне введите имя класса FormatCommands и нажмите кнопку Add или клавишу Enter. Будет создана заготовка файла FormatCommands.cs, в которую надо ввести новые операторы. Приведем окончательный вид файла FormatCommands.cs, в котором полужирным шрифтом выделены добавленные операторы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace TEXTEDIT
{
class FormatCommands
{
    private static RoutedUICommand fontcol;
    private static RoutedUICommand backcol;
    static FormatCommands()
    {
        InputGestureCollection inputs = new
          InputGestureCollection();
        inputs.Add(new KeyGesture(Key.F, ModifierKeys.Control |
          ModifierKeys.Shift, "Ctrl+Shift+F"));
        fontcol = new RoutedUICommand("Font Color...",
          "FontColor", typeof(FormatCommands), inputs);
        inputs = new InputGestureCollection();
        inputs.Add(new KeyGesture(Key.B, ModifierKeys.Control |
```

```
            ModifierKeys.Shift, "Ctrl+Shift+B"));
        backcol = new RoutedUICommand("Back Color...",
            "BackColor", typeof(FormatCommands), inputs);
    }
    public static RoutedUICommand FontColor
    {
        get { return fontcol; }
    }
    public static RoutedUICommand BackColor
    {
        get { return backcol; }
    }
}
}
```

При определении команды указывается ее подпись, имя, тип и набор клавиш быстрого доступа (а при определении этого набора также задается и текст, связанный с клавишами).

После этого необходимо *зарегистрировать* новые команды в окне нашей программы и связать их с обработчиками. Это делается в xaml-файле в элементе Window.CommandBindings:

```
<Window.CommandBindings>
  …
  <CommandBinding x:Name="fontColor0"
      Command="local:FormatCommands.FontColor"
      Executed="fontColor0_Executed" />
  <CommandBinding x:Name="backColor0"
      Command="local:FormatCommands.BackColor"
      Executed="backColor0_Executed" />
</Window.CommandBindings>
```

Обратите внимание на префикс local, указанный перед именами команд. Этот префикс был ранее определен в xaml-файле следующим образом:

```
xmlns:local="clr-namespace:TEXTEDIT"
```

Он позволяет указать *пространство имен*, в котором следует искать указанные классы. Если в xaml-файле требуется использовать другие нестандартные пространства имен, то для них тоже можно определить префиксы.

Заметим, что при описанном выше дополнении xaml-файла текст атрибутов Command может быть выделен как ошибочный (синим подчеркиванием), однако это не будет препятствовать успешной компиляции проекта. После такой компиляции подчеркивание должно исчезнуть.

К списку пунктов меню надо добавить следующие элементы:

```
<MenuItem x:Name="format1" Header="_Format" >
  <MenuItem x:Name="bold1" Header="_Bold" … />
  …
  <MenuItem x:Name="rightJustify1" … />
  <Separator/>
  <MenuItem x:Name="color1" Header="_Color" >
    <MenuItem x:Name="fontColor1"
        Command="local:FormatCommands.FontColor" />
    <MenuItem x:Name="backColor1"
        Command="local:FormatCommands.BackColor" />
  </MenuItem>
</MenuItem>
```

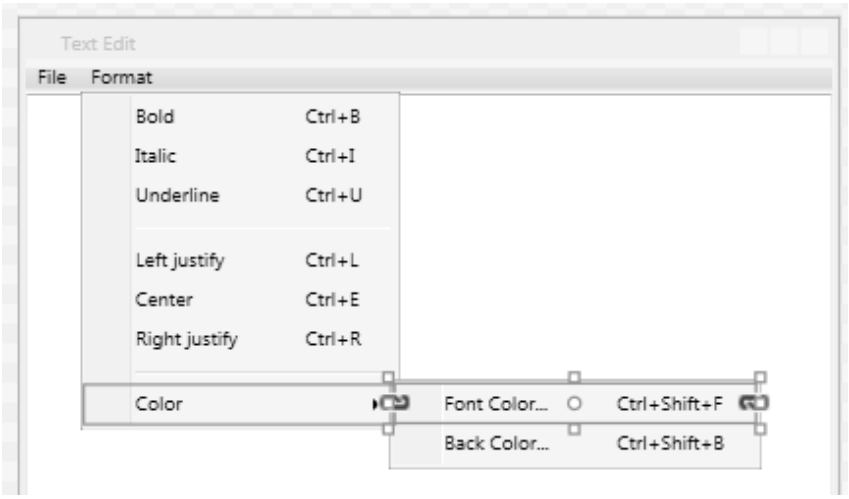

Рис. 2. Макет окна приложения TEXTEDIT версии 2 (окончательный вариант)

Таким образом, мы создали меню третьего уровня (см. рис. 36). Поскольку все настройки сделаны в самих командах WPF, при определении пунктов меню было достаточно только указать эти команды.

Осталось обратиться к файлу MainWindow.xaml.cs и определить обработчики для новых команд. Однако здесь нас ожидают дополнительные проблемы.

Дело в том, что в библиотеке WPF отсутствует не только компонент, обеспечивающий вызов диалогового окна для выбора цвета, но и обычный класс для выполнения такого действия. Поэтому, если мы не хотим вручную создавать подобное диалоговое окно, придется воспользоваться диалоговым окном выбора цвета из библиотеки Windows Forms.

Подключите к проекту необходимые компоненты библиотеки Windows Forms (System.Windows.Forms и System.Drawing), действуя так, как описано в п. 8.5 проекта CURSORS. Напомним, что указывать соответствующие пространства имен в директивах using не следует, чтобы избе-

жать конфликтов совпадающих имен для классов из библиотек Windows Forms и WPF.

Теперь в нашей программе можно использовать класс ColorDialog из библиотеки Windows Forms. Но перед этим надо решить еще одну проблему, связанную с тем, что этот класс работает с классом Color из библиотеки Windows Forms, а компоненты WPF – с классом Color из библиотеки WPF, и эти классы несовместимы. Поэтому в классе MainWindow необходимо реализовать два вспомогательных метода-*конвертера*, преобразующих один класс Color в другой. Эти методы выглядят следующим образом:

```
Color WinFormsColorToWPFColor(System.Drawing.Color c)
{
    return Color.FromArgb(c.A, c.R, c.G, c.B);
}

System.Drawing.Color WPFColorToWinFormsColor(Color c)
{
    return System.Drawing.Color.FromArgb(c.A, c.R, c.G, c.B);
}
```

Осталось добавить в класс MainWindow новое поле

```
System.Windows.Forms.ColorDialog colorDialog1 =
  new System.Windows.Forms.ColorDialog();
```

и определить обработчики событий для новых команд (заготовки этих обработчиков уже содержатся в описании класса MainWindow):

```
private void fontColor0_Executed(object sender,
  ExecutedRoutedEventArgs e)
{
    colorDialog1.Color =
      WPFColorToWinFormsColor((textBox1.Foreground as
      SolidColorBrush).Color);
    if (colorDialog1.ShowDialog() ==
      System.Windows.Forms.DialogResult.OK)
        textBox1.Foreground = new
          SolidColorBrush(WinFormsColorToWPFColor(colorDialog1
          .Color));
}
private void backColor0_Executed(object sender,
  ExecutedRoutedEventArgs e)
{
    colorDialog1.Color =
      WPFColorToWinFormsColor((textBox1.Background as
```

```
      SolidColorBrush).Color);
    if (colorDialog1.ShowDialog() ==
      System.Windows.Forms.DialogResult.OK)
        textBox1.Background = new
          SolidColorBrush(WinFormsColorToWPFColor(colorDialog1
          .Color));
}
```

Результат. При выполнении команды из группы Colors вызывается диалоговое окно «Цвет», позволяющее установить цвет символов (команда Font color) или цвет фона (команда Background color). При отображении диалогового окна в нем сплошной рамкой обводится текущий цвет. При выборе нового цвета и нажатии кнопки OK происходит изменение цвета в редакторе.

2. Команды редактирования и контекстное меню: TEXTEDIT, версия 3

2.1. Команды редактирования

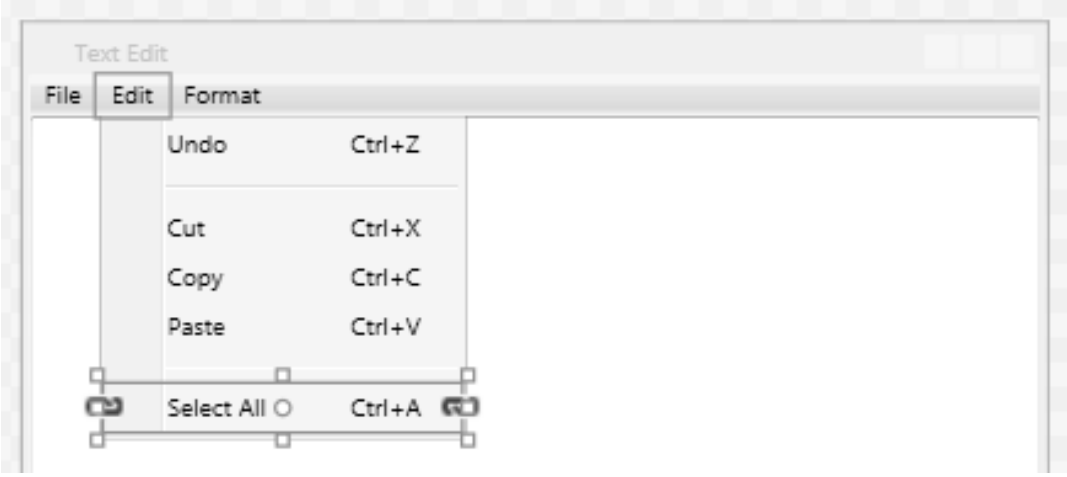

Рис. 3. Макет окна приложения TEXTEDIT версии 3

```
<Window x:Class="TEXTEDIT.MainWindow"
    … >
  <Window.CommandBindings>
    …
    <CommandBinding x:Name="undo0"
        Command="ApplicationCommands.Undo"
        Executed="undo0_Executed"
        CanExecute="undo0_CanExecute" />
    <CommandBinding x:Name="cut0"
        Command="ApplicationCommands.Cut"
        Executed="cut0_Executed" CanExecute="cut0_CanExecute" />
    <CommandBinding x:Name="copy0"
        Command="ApplicationCommands.Copy"
        Executed="copy0_Executed" CanExecute="cut0_CanExecute" />
    <CommandBinding x:Name="paste0"
        Command="ApplicationCommands.Paste"
        Executed="paste0_Executed"
        CanExecute="paste0_CanExecute" />
    <CommandBinding x:Name="selectAll0"
        Command="ApplicationCommands.SelectAll"
        Executed="selectAll0_Executed"
```

```
        CanExecute="selectAll0 CanExecute" />
  </Window.CommandBindings>
  <DockPanel >
    <Menu DockPanel.Dock="Top" >
      <MenuItem x:Name="file1" Header="_File" >
        …
      </MenuItem>
      <MenuItem x:Name="edit1" Header="_Edit" >
        <MenuItem x:Name="undo1" Header="_Undo"
            Command="ApplicationCommands.Undo" />
        <Separator/>
        <MenuItem x:Name="cut1" Header="Cu_t"
            Command="ApplicationCommands.Cut"/>
        <MenuItem x:Name="copy1" Header="_Copy"
            Command="ApplicationCommands.Copy"/>
        <MenuItem x:Name="paste1" Header="_Paste"
            Command="ApplicationCommands.Paste" />
        <Separator/>
        <MenuItem x:Name="selectAll1" Header="_Select All"
            Command="ApplicationCommands.SelectAll"/>
      </MenuItem>
      <MenuItem x:Name="format1" Header="_Format" >
        …
      </MenuItem>
    </Menu>
    …
  </DockPanel>
</Window>
```

При задании обработчиков событий для команд WPF в xaml-файле обратите внимание на то, что для команд cut0 и copy0 используется *один и тот же обработчик* для события CanExecuted: cut0_CanExecute.

Определите в классе MainWindow новые обработчики, указанные в xaml-файле:

```
private void undo0_Executed(object sender,
  ExecutedRoutedEventArgs e)
{
    textBox1.Undo();
}
private void cut0_Executed(object sender,
  ExecutedRoutedEventArgs e)
{
```

```
    textBox1.Cut();
}
private void copy0_Executed(object sender,
  ExecutedRoutedEventArgs e)
{
    textBox1.Copy();
}
private void paste0_Executed(object sender,
  ExecutedRoutedEventArgs e)
{
    textBox1.Paste();
}
private void selectAll0_Executed(object sender,
  ExecutedRoutedEventArgs e)
{
    textBox1.SelectAll();
}
private void undo0_CanExecute(object sender,
  CanExecuteRoutedEventArgs e)
{
    e.CanExecute = textBox1.CanUndo;
}
private void cut0_CanExecute(object sender,
  CanExecuteRoutedEventArgs e)
{
    e.CanExecute = textBox1.SelectedText.Length > 0;
}
private void paste0_CanExecute(object sender,
  CanExecuteRoutedEventArgs e)
{
    e.CanExecute = Clipboard.ContainsText();
}
private void selectAll0_CanExecute(object sender,
  CanExecuteRoutedEventArgs e)
{
    e.CanExecute = textBox1.Text.Length > 0;
}
```

Результат. В меню определены стандартные *команды редактирования*: отмена последнего действия (Undo), вырезание (Cut) и копирование (Copy) выделенного фрагмента в буфер, вставка фрагмента из буфера

(Paste), выделение всего текста (Select All). При вызове подменю группы Edit недоступные команды выделяются серым цветом.

Недочет. Эксперименты, проведенные с программой, показывают, что обработчик selectAll0_CanExecute *никогда не вызывается*. Соответственно, команда Select All всегда доступна (даже в случае, когда компонент TextBox не содержит никакого текста). Видимо, разработчики WPF решили, что невозможны ситуации, когда команда выделения всех данных должна быть недоступной. Можно или примириться с этим обстоятельством, или определить собственную команду WPF для выделения всех данных, которая будет обрабатывать событие CanExecute. В случае выбора второго варианта действий его реализацию мы предоставляем читателю.

2.2. Создание контекстного меню

Для подключения к полю ввода контекстного меню проще всего выбрать в окне Properties для компонента TextBox свойство ContextMenu и нажать находящуюся рядом с ним кнопку New. В xaml-файле будет создана следующая заготовка для этого меню:

```
<TextBox x:Name="textBox1" … >
  <TextBox.ContextMenu>
    <ContextMenu />
  </TextBox.ContextMenu>
</TextBox>
```

После этого надо добавить пункты меню в качестве дочерних элементов элемента ContextMenu (как мы это делали ранее для основного меню). Напомним, что для превращения комбинированного тега <ContextMenu /> в парный достаточно стереть символы «/>» в конце этого тега и повторно ввести символ «>».

```
<TextBox x:Name="textBox1" >
  <TextBox.ContextMenu>
    <ContextMenu>
      <MenuItem Header="Cu_t" Command="ApplicationCommands.Cut"/>
      <MenuItem Header="_Copy"
          Command="ApplicationCommands.Copy"/>
      <MenuItem Header="_Paste"
          Command="ApplicationCommands.Paste"/>
      <Separator/>
      <MenuItem Header="_Select All"
          Command="ApplicationCommands.SelectAll"/>
      <Separator/>
      <MenuItem Command="local:FormatCommands.FontColor"/>
      <MenuItem Command="local:FormatCommands.BackColor"/>
```

```
    </ContextMenu>
  </TextBox.ContextMenu>
</TextBox>
```

Результат. При нажатии правой кнопки мыши в области редактирования или комбинации Shift+F10 вместо стандартного контекстного меню компонента TextBox (содержащего команды для работы с буфером «Вырезать», «Копировать», «Вставить») отображается контекстное меню, определенное в свойстве ContextMenu данного компонента. Недоступные команды контекстного меню выделяются серым цветом.

Таким образом, мы реализовали возможность использования собственного контекстного меню, не написав ни одной строки программного кода.