Глава 21. Панель инструментов: проект TXTEDIT4

21.1. Создание панели инструментов с кнопками быстрого доступа. Добавление изображений к пунктам меню

В качестве заготовки для проекта TXTEDIT4 следует использовать ранее разработанный проект TXTEDIT3 (см. гл. 20).

Разместите в форме `Form1` *панель инструментов* — компонент типа `ToolStrip` (этот компонент получит имя `toolStrip1`). Панель инструментов автоматически пристыкуется к верхней границе формы и будет располагаться ниже главного меню `menuStrip1`. Однако она будет заслонять верхнюю часть компонента `textBox1`. Для размещения всех трех компонентов без перекрытий необходимо установить для них правильный *z*-порядок. В нашем случае проще всего выполнить команду **Bring to Front** для компонента `textBox1`. Заметим, что для этой команды (как и для команды **Send To Back**) предусмотрена не только соответствующая кнопка на панели **Layout**, но и пункт контекстного меню, которое можно вызвать для любого визуального компонента, размещенного в форме.

Помещенная в форму панель инструментов содержит *выпадающий список всех типов компонентов*, которые можно разместить на этой панели. В нашей программе будут использоваться только обычные кнопки (**Button**) и разделители (**Separator**). При выборе варианта **Button** компонент-кнопка типа `ToolStripButton` размещается на панели и ему сразу присваивается имя (в данном случае `toolStripButton1`). Поскольку кнопки панели инструментов *(кнопки быстрого доступа)* будут связаны с соответствующими командами главного меню, мы будем заменять имя кнопки, предлагаемое по умолчанию, на имя соответствующего пункта меню, дополняя его цифрой `0`, например, `new10`.

Помещенные на панель компоненты автоматически выстраиваются в ряд без промежутков между ними. Для добавления стандартного промежутка-разделителя следует выбрать из списка возможных элементов панели вариант **Separator**. При этом на панель добавляется компонент типа `ToolStripSeparator`, и ему сразу присваивается соответствующее имя (например, `toolStripSeparator1`). Так как в дальнейшем к разделителям обращаться не требуется, мы не будем изменять их имена. Заметим, что взаимное расположение кнопок и разделителей на панели можно изменить, перетаскивая мышью требуемый элемент на новое место.

Добавьте на панель инструментов `toolStrip1` следующие компоненты (в порядке слева направо; имена компонентов-кнопок, то есть их свойства `Name`, следует указывать в окне **Properties**): кнопка **`new10`**, кнопка **`open10`**, кнопка **`save10`**, разделитель, кнопка **`cut10`**, кнопка **`copy10`**, кнопка **`paste10`**.

Свойство `Text` кнопки быстрого доступа по умолчанию совпадает с ее именем (то есть свойством `Name`), однако на кнопке оно не отображается, так как предполагается, что кнопка будет содержать не текст, а изображение (за внешний вид кнопки отвечает ее свойство `DisplayStyle`, имеющее по умолчанию значение **`Image`**). Со всеми кнопками, добавленными на панель инструментов, связывается стандартное изображение . Разумеется, его надо заменить на изображение, соответствующее действию, которое должна выполнять каждая кнопка.

Перед назначением кнопкам изображений следует добавить все необходимые изображения в файл ресурсов разрабатываемого проекта. Опишем действия по добавлению изображений, предполагая, что нам доступна коллекция ресурсов Visual Studio 2005 или 2008.

Выделите первую из добавленных на панель кнопок (`new10`) и в окне **Properties** перейдите к свойству `Image`. Нажмите на кнопку с многоточием рядом с этим свойством, в появившемся окне **Select Resource** выберите переключатель **Project resource file** и нажмите кнопку **Import...**. В появившемся окне **Open** перейдите в подкаталог bitmaps\commands\24bitcolor коллекции VS2005ImageLibrary (или подкаталог Actions\24bitcolor bitmaps коллекции VS2008ImageLibrary), выберите файл Document.bmp и нажмите кнопку **Открыть** или клавишу <Enter>. В результате в окне **Select Resource** появится имя добавленного ресурса: **Document**. Не закрывая окна **Select Resource**, добавьте в файл ресурсов следующие файлы из того же подкаталога: OpenFolder.bmp, Save.bmp, Cut.bmp, Copy.bmp, Paste.bmp.

После добавления всех изображений выберите из их списка вариант **Document** и нажмите кнопку **OK**. Выбранное изображение будет связано с кнопкой быстрого доступа `new10`. Аналогичными действиями следует добавить изображения ко всем кнопкам быстрого доступа. В листинге 21.1 для каждой кнопки указывается имя того изображения из списка ресурсов, которое надо указать в свойстве `Image`. Кроме того, в листинге 21.1 указано, с каким существующим обработчиком надо связать свойство `Click` каждой кнопки. Еще одно свойство, которое следует настроить для каждой кнопки — это `ToolTipText`; данное свойство отвечает за отображение всплывающей подсказки при наведении курсора мыши на кнопку.

Проверьте, что для всех кнопок быстрого доступа свойство `ImageTransparentColor` имеет значение **`Magenta`** (этим цветом во всех использованных изображениях окрашены фрагменты, которые должны быть прозрачными). Вид полученной панели инструментов приводится на рис. 21.1.

После включения изображений в файл ресурсов проекта их можно использовать не только для кнопок быстрого доступа, но и для пунктов главного меню. Для этого достаточно связать соответствующее изображение со свойством `Image` пункта меню (действуя так же, как при связывании изображения с кнопкой). Приведем для каждого пункта меню имя изображения, которое надо с ним связать: `new1` — **`Document`**, `open1` — **`OpenFolder`**, `save1` — **`Save`**, `cut1` — **`Cut`**, `copy1` — **`Copy`**, `paste1` — **`Paste`**. Для всех этих пунктов меню необходимо установить значение свойства `ImageTransparentColor` равным **`Magenta`**.

Рис. 21.1. Верхняя часть формы Form1 для проекта TXTEDIT4 на начальном этапе разработки

Листинг 21.1. Настройка свойств

```
new10: Image = Document, ToolTipText = New,
  Click = new1_Click
open10: Image = OpenFolder, ToolTipText = Open,
  Click = open1_Click
save10: Image = Save, ToolTipText = Save,
  Click = save1_Click
cut10: Image = Cut, ToolTipText = Cut,
  Click = cut1_Click
copy10: Image = Copy, ToolTipText = Copy,
  Click = copy1_Click
paste10: Image = Paste, ToolTipText = Paste,
  Click = paste1_Click
```

Результат: для выполнения часто используемых команд теперь достаточно щелкнуть мышью на соответствующей кнопке быстрого доступа. Чтобы определить, с какой командой связана кнопка быстрого доступа, надо переместить на нее курсор мыши: через 1–2 секунды около кнопки появится желтый ярлычок с названием соответствующей команды. Около команд меню, имеющих кнопки быстрого доступа, изображаются те же картинки, что и на связанных с ними кнопках.

21.2. Размещение на панели инструментов кнопок-флажков и кнопок-переключателей

Для того чтобы рассмотреть особенности, связанные с выводом на кнопках быстрого доступа не изображений, а текста, все кнопки, добавленные на панель инструментов в данном разделе, будут снабжены *текстовыми подписями*. Эти подписи указываются в свойстве `Text`. Для отображения на кнопке текстовой подписи свойство `DisplayStyle` кнопки необходимо положить равным **`Text`**.

Добавьте на панель инструментов `toolStrip1` новые компоненты (в порядке слева направо; имена компонентов-кнопок, то есть их свойства `Name`, следует указывать в окне **Properties**): разделитель, кнопка **`bold10`**, кнопка **`italic10`**, кнопка **`underline10`**, разделитель, кнопка **`leftJustify10`**, кнопка **`center10`**, кнопка **`rightJustify10`**. Настройте свойства добавленных компонентов (листинг 21.2). Вид полученной панели инструментов приводится на рис. 21.2.

В класс `Form1` добавьте вспомогательный метод `GetButton` (листинг 21.3) и определите обработчики события `Click` для новых кнопок быстрого доступа (листинг 21.4).

Измените методы `bold1_Click`, `leftJustify1_Click` и `font1_Click` (листинг 21.5).

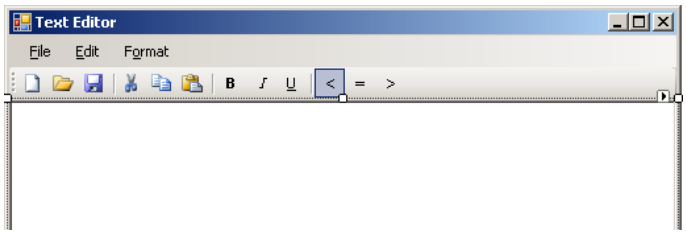

Рис. 21.2. Окончательный вариант верхней части формы Form1 для проекта TXTEDIT4

Листинг 21.2. Настройка свойств

```
bold10: Text = B, DisplayStyle = Text,
  ToolTipText = Bold, Font.Bold = True
italic10: Text = I, DisplayStyle = Text,
  ToolTipText = Italic, Font.Italic = True
underline10: Text = U, DisplayStyle = Text,
  ToolTipText = Underline, Font.Underline = True
leftJustify10: Text = <, DisplayStyle = Text,
  ToolTipText = Left justify, Checked = True
center10: Text = =, DisplayStyle = Text,
  ToolTipText = Center
rightJustify10: Text = >, DisplayStyle = Text,
  ToolTipText = Right justify
```

Листинг 21.3. Метод `GetButton` формы `Form1`

```
private ToolStripButton GetButton(ToolStripMenuItem mi)
{
```

```
  return toolStrip1.Items[mi.Name + "0"] as ToolStripButton;
}
```

Листинг 21.4. Обработчики `bold10.Click`, `italic10.Click`, `underline10.Click`, `leftJustify10.Click`, `center10.Click`, `rightJustify10.Click`

```
private void bold10_Click(object sender, EventArgs e)
{
  bold1_Click(bold1, null);
}
private void italic10_Click(object sender, EventArgs e)
{
  bold1_Click(italic1, null);
}
private void underline10_Click(object sender, EventArgs e)
{
  bold1_Click(underline1, null);
}
private void leftJustify10_Click(object sender, EventArgs e)
{
  leftJustify1_Click(leftJustify1, null);
}
private void center10_Click(object sender, EventArgs e)
{
  leftJustify1_Click(center1, null);
}
private void rightJustify10_Click(object sender, EventArgs e)
{
  leftJustify1_Click(rightJustify1, null);
}
```

Листинг 21.5. Новый вариант методов `bold1_Click`, `leftJustify1_Click` и `font1_Click`

```
private void bold1_Click(object sender, EventArgs e)
{
  ToolStripMenuItem mi = sender as ToolStripMenuItem;
  mi.Checked = !mi.Checked;
  FontStyle fs = textBox1.Font.Style;
  fs = mi.Checked ? (fs | mi.Font.Style) : (fs & ~mi.Font.Style);
  Font f = textBox1.Font;
  textBox1.Font = new Font(f, fs);
  f.Dispose();
  ToolStripButton sb = GetButton(mi);
  sb.Checked = !sb.Checked;
}
private void leftJustify1_Click(object sender, EventArgs e)
{
  ToolStripMenuItem mi = sender as ToolStripMenuItem;
  if (mi.Checked) return;
  GetButton(alignItem).Checked = alignItem.Checked = false;
  alignItem = mi;
  mi.CheckState = CheckState.Indeterminate;
  GetButton(mi).Checked = true;
  textBox1.TextAlign = (HorizontalAlignment)mi.Tag;
}
private void font1_Click(object sender, EventArgs e)
{
  fontDialog1.Font = textBox1.Font;
  if (fontDialog1.ShowDialog() == DialogResult.OK)
    if (!textBox1.Font.Equals(fontDialog1.Font))
    {
      Font f = textBox1.Font;
      textBox1.Font = fontDialog1.Font;
      f.Dispose();
      bold10.Checked = bold1.Checked = fontDialog1.Font.Bold;
      italic10.Checked = italic1.Checked = fontDialog1.Font.Italic;
      underline10.Checked = underline1.Checked = fontDialog1.Font.Underline;
    }
}
```

Результат: для настройки стиля шрифта и выравнивания текста достаточно нажать на соответствующую кнопку быстрого доступа, переведя ее тем самым в "нажатое" состояние. Далее мы будем говорить о "нажатом" и "отжатом" состоянии кнопки, хотя при использовании стандартного стиля изображения кнопок в .NET "нажатая" кнопка просто выделяется рамкой и фоновым цветом (см. изображение кнопки < на рис. 21.2).

Нажатое состояние кнопки быстрого доступа означает, что указанный режим включен. Кнопки настройки шрифта `bold10`, `italic10`, `underline10` действуют как *флажки*: каждую кнопку можно перевести в нажатое или отжатое состояние неза-

висимо от остальных. Кнопки выравнивания текста `leftJustify10`, `center10`, `rightJustify10` действуют как *группа переключателей*: нажатие на любую из них приводит к освобождению (отжатию) остальных. Следует отметить, что при выполнении команд форматирования непосредственно из меню или с помощью клавиш-ускорителей состояние соответствующих кнопок быстрого доступа корректируется требуемым образом.

Глава 22. Статусная панель и подсказки: проект TXTEDIT5

22.1. Использование статусной панели

В качестве заготовки для проекта TXTEDIT5 следует использовать ранее разработанный проект TXTEDIT4 (см. гл. 21).

Добавьте в форму `Form1` *статусную панель* — компонент типа `StatusStrip` (этот компонент получит имя `statusStrip1`), а также невизуальный компонент типа `Timer` (который получит имя `timer1` и будет указан в области невизуальных компонентов под изображением формы). Для того чтобы нижняя часть компонента `textBox1` не была перекрыта добавленной статусной панелью, следует выполнить команду **Bring to Front**  для компонента `textBox1` (см. разд. 21.1).

Для добавления компонентов на статусную панель предназначен выпадающий список, в котором мы будем использовать только вариант **StatusLabel** (этот вариант добавляет на статусную панель компонент-метку типа `ToolStripStatusLabel`). Поскольку имена по умолчанию для добавленных на панель компонентов являются чрезмерно длинными (например, `toolStripStatusLabel1`), мы будем изменять их, выбирая имена в соответствии с назначением каждого элемента статусной строки.

Добавьте на статусную панель `statusStrip1` компоненты-метки с указанными именами (в порядке слева направо; имена компонентов-меток, то есть их свойства `Name`, следует указывать в окне **Properties**): **`cap1`**, **`num1`**, **`modified1`**, **`hint1`**. Настройте свойства добавленных компонентов-меток, а также компонента `timer1` (листинг 22.1; свойство `Size.Width` используется для задания новой ширины метки, а свойство `Margin.Left` метки `hint1` — для увеличения промежутка между текстом **Ready** этой метки и рамкой предыдущей метки `modified1`). Вид полученной статусной панели приводится на рис. 22.1.

Определите обработчик события `Tick` для компонента `timer1` (листинг 22.2) и добавьте вызов этого обработчика в конструктор класса `Form1`:

```
timer1_Tick(this, null);
```

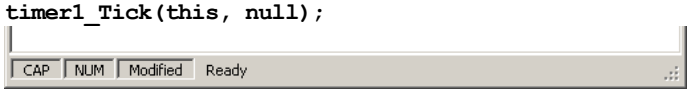

Рис. 22.1. Вид нижней части Form1 для проекта TXTEDIT5

Листинг 22.1. Настройка свойств

```
cap1: Text = CAP, AutoSize = False,
  BorderSides = All, BorderStyle = Sunken,
  Size.Width = 40
num1: Text = NUM, AutoSize = False,
  BorderSides = All, BorderStyle = Sunken,
  Size.Width = 40
modified1: Text = Modified, AutoSize = False,
  BorderSides = All, BorderStyle = Sunken,
  Size.Width = 60
hint1: Text = Ready, Margin.Left = 5
timer1: Interval = 200, Enabled = True
```

Листинг 22.2. Обработчик `timer1.Tick`

```
private void timer1_Tick(object sender, EventArgs e)
{
  cap1.Text = Control.IsKeyLocked(Keys.CapsLock) ? "CAP" : "";
  num1.Text = Control.IsKeyLocked(Keys.NumLock) ? "NUM" : "";
  modified1.Text = textBox1.Modified ? "Modified" : "";
}
```

Результат: на статусной панели отображается текущее состояние клавиш <Caps Lock> и <Num Lock>. Кроме того, в ней показывается, изменялся ли документ после его последнего сохранения (если изменялся, то на панели выводится строка **Modified**).

22.2. Недоступные кнопки быстрого доступа

Добавьте в метод `timer1_Tick` новые операторы (листинг 22.3).

Листинг 22.3. Добавление к методу `timer1_Tick`

```
cut10.Enabled = copy10.Enabled = textBox1.SelectionLength > 0;
paste10.Enabled =
  Clipboard.GetDataObject().GetDataPresent(typeof(string));
```

```
save1.Enabled = save10.Enabled = textBox1.Modified;
```

Результат: теперь состояние кнопок быстрого доступа, связанных с буфером обмена, совпадает с состоянием соответствующих команд меню (они одновременно доступны или недоступны). Кроме того, команда **Save** и связанная с ней кнопка быстрого доступа теперь доступны только после внесения изменений в редактируемый текст.

22.3. Скрытие панелей

Действуя так же, как в разд. 18.1 при создании пункта меню **File** и связанных с ним пунктов меню второго уровня, добавьте в компонент `menuStrip1` пункт меню первого уровня с текстом **`&View`** и с помощью окна **Properties** измените имя этого пункта (то есть свойство `Name`) на **`view1`**. В выпадающем меню, связанном с пунктом **View**, создайте пункты с текстом **`&Tool bar`** и **`&Status bar`** (рис. 22.2). Настройте свойства добавленных пунктов меню (листинг 22.4).

Определите обработчики события `Click` для пунктов меню `toolBar1` и `statusBar1` (листинг 22.5).

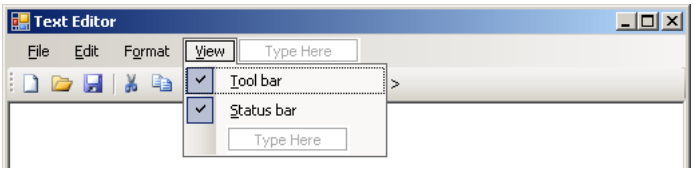

Рис. 22.2. Вид верхней части Form1 для проекта TXTEDIT5

Листинг 22.4. Настройка свойств

```
пункт меню Tool bar (группа View): Name = toolBar1,
  Checked = True
пункт меню Status bar (группа View):
  Name = statusBar1, Checked = True
```

Листинг 22.5. Обработчики `toolbar1.Click` и `statusBar1.Click`

```
private void toolBar1_Click(object sender, EventArgs e)
{
  toolStrip1.Visible = toolBar1.Checked = !toolBar1.Checked;
}
private void statusBar1_Click(object sender, EventArgs e)
{
  statusStrip1.Visible = statusBar1.Checked = !statusBar1.Checked;
}
```

Результат: с помощью команд-переключателей меню **View** можно скрывать и восстанавливать панель инструментов и статусную панель.

22.4. Вывод подсказок на статусную панель

Настройте свойство `ToolTipText` для пунктов меню группы **File** (листинг 22.6).

Определите обработчики событий `MouseEnter` и `MouseLeave` для пункта меню `new1` (листинг 22.7), после чего свяжите эти обработчики с событиями `MouseEnter` и `MouseLeave` остальных пунктов меню группы **File**. Обработчик `new1_MouseLeave` свяжите также с событием `MenuActivate` компонента `menuStrip1`.

Кроме того, определите обработчик события `MenuDeactivate` для компонента `menuStrip1` (листинг 22.8).

Листинг 22.6. Настройка свойств

```
new1: ToolTipText = Создать новый документ
open1: ToolTipText = Открыть существующий документ
save1: ToolTipText = Сохранить текущий документ
saveAs1: ToolTipText = Сохранить документ под новым именем
exit1: ToolTipText = Выйти из редактора
```

Листинг 22.7. Обработчики `new1.MouseEnter` и `new1.MouseLeave`

```
private void new1_MouseEnter(object sender, EventArgs e)
{
  hint1.Text = (sender as ToolStripMenuItem).ToolTipText;
}
private void new1_MouseLeave(object sender, EventArgs e)
{
  hint1.Text = "";
}
```

Листинг 22.8. Обработчик `menuStrip1.MenuDeactivate`

```
private void menuStrip1_MenuDeactivate(object sender, EventArgs e)
{
  hint1.Text = "Ready";
}
```

Результат: при перемещении курсора мыши над пунктами меню **File** на статусной панели выводится подсказка к выделенной команде (если данная команда является доступной). Для прочих команд меню подсказка на статусной панели не отображается. При выходе из меню на статусной панели восстанавливается текст **Ready**.

Недочет: при наведении мыши на пункт меню из группы **File** около данного пункта возникает дополнительная всплывающая подсказка, текст которой совпадает с текстом, выведенным на статусную панель.

Исправление: в конструктор класса `Form1` добавьте оператор, подавляющий отображение всплывающих подсказок для выпадающего меню **File**:

```
file1.DropDown.ShowItemToolTips = false;
```

Примечание

Настроить свойство `ShowItemToolTips` для *главного меню* можно в окне свойств для компонента `menuStrip1` (заметим, что по умолчанию это свойство равно **`False`**). Однако настроить его для *выпадающих подменю* (объектов типа `ToolStripDropDown`) можно только программным путем.

Глава 23. Форматирование документа: проект TXTEDIT6

23.1. Замена компонента *TextBox* на компонент *RichTextBox*

В качестве заготовки для проекта TXTEDIT6 следует использовать ранее разработанный проект TXTEDIT5 (см. гл. 22).

Для того чтобы заменить уже имеющийся в форме `Form1` компонент `textBox1` (типа `TextBox`) на компонент типа `RichTextBox`, обладающий более богатыми возможностями форматирования, достаточно немного откорректировать файл Form1.Designer.cs. Напомним, что в этом файле содержится информация, добавленная в проект в ходе работы с дизайнером формы (в том числе с дизайнером меню) и окном **Properties**, поэтому обычно не возникает необходимости в его явном редактировании. Однако ничто не запрещает вносить в файл Form1.Designer.cs изменения; следует только учитывать, что именно из этого файла берутся сведения, используемые для отображения формы в режиме дизайна.

Загрузите в редактор файл Form1.Designer.cs (проще всего это сделать с помощью окна **Solution Explorer**), найдите в нем строку

```
private System.Windows.Forms.TextBox textBox1;
```

и измените ее следующим образом:

```
private System.Windows.Forms.RichTextBox textBox1;
```

Тем самым мы изменили тип компонента `textBox1`. Имя компонента изменять не будем, так как оно неоднократно используется в имеющемся коде нашей программы.

Необходимо также изменить оператор, который создает компонент `textBox1`. Данный оператор содержится в том же файле Form1.Designer.cs, причем в разделе **Windows Form Designer generated code**, который по умолчанию является скрытым. Разверните этот раздел, щелкнув мышью на знаке **+** слева от его заголовка, и найдите в нем оператор

```
this.textBox1 = new System.Windows.Forms.TextBox();
```

Для нахождения указанного оператора можно, например, организовать поиск слова **textBox1**. Измените найденный оператор следующим образом:

```
this.textBox1 = new System.Windows.Forms.RichTextBox();
```

После внесения указанных изменений файл Form1.Designer.cs можно закрыть, щелкнув правой кнопкой мыши на его ярлычке и выбрав в появившемся контекстном меню команду **Close**.

Компонент `RichTextBox` позволяет устанавливать различные форматные настройки для разных фрагментов текста. Эти настройки сохраняются вместе с текстом в специальном формате, называемом *Rich Text Format* (формат RTF). Файлы, содержащие текст в данном формате, обычно имеют расширение rtf. Внесите в программу изменения, позволяющие использовать дополнительные возможности компонента `RichTextBox`.

Для компонентов `saveFileDialog1` и `openFileDialog1` измените значения свойства `DefaultExt` на **`rtf`**, а свойства `Filter` на **`RTF files|*.rtf`**.

Измените методы `SaveToFile` и `open1_Click` класса `Form1` (листинг 23.1).

В методах `bold1_Click` и `font1_Click` (листинг 21.5) замените все фрагменты `textBox1.Font` на **`textBox1.SelectionFont`** (в `bold1_Click` таких фрагментов должно быть три, в `font1_Click` — четыре).

В методе `fontColor1_Click` (листинг 19.7) замените все фрагменты `textBox1.ForeColor` на **`textBox1.SelectionColor`** (таких фрагментов должно быть два).

В методе `backgroundColor1_Click` (листинг 19.7) замените все фрагменты `textBox1.BackColor` на **`textBox1.SelectionBackColor`** (таких фрагментов должно быть два).

В методе `leftJustify1_Click` (листинг 21.5) замените в последнем операторе фрагмент `textBox1.TextAlign` на **`textBox1.SelectionAlignment`**.

Наконец, установите свойство `EnableAutoDragDrop` компонента `textBox1` равным **`True`**.

Листинг 23.1. Новый вариант методов `SaveToFile` и `open1_Click`

```
private void SaveToFile(string path)
{
```

```
  textBox1.SaveFile(path, RichTextBoxStreamType.RichText);
  textBox1.Modified = false;
}
private void open1_Click(object sender, EventArgs e)
{
  if (TextSaved())
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
      string path = openFileDialog1.FileName;
      textBox1.LoadFile(path, RichTextBoxStreamType.RichText);
      Text = "Text Editor - " + Path.GetFileName(path);
      saveFileDialog1.FileName = path;
      openFileDialog1.FileName = "";
    }
}
```

Результат: теперь команды форматирования шрифта влияют на выделенный фрагмент текста, а если выделенного фрагмента нет, то на последующие вводимые символы. Команды выравнивания абзаца влияют на выделенные абзацы (в том числе и на абзацы, в которых выделена только часть текста), а если выделенных абзацев нет, то на текущий абзац. В файле можно сохранять текст вместе с форматными настройками (расширение файла по умолчанию — rtf). При загрузке файлов в формате RTF на экране отображается текст вместе с сохраненными форматными настройками.

Если текст не умещается по высоте в окне редактора, то справа появляется вертикальная полоса прокрутки (подобная возможность *автоматического* отображения вертикальной полосы прокрутки у компонента `TextBox` отсутствует). Слишком большие по ширине строки, как и ранее, автоматически переносятся на новую строку.

Благодаря значению **`True`** свойства `EnableAutoDragDrop`, появившегося у компонента `RichTextBox` в версии .NET 2.0, в редакторе обеспечивается режим перетаскивания drag & drop для выделенных фрагментов текста: если зацепить мышью выделенный фрагмент, то его можно перетащить на новое место, а если при перетаскивании держать нажатой клавишу <Ctrl> (при этом около курсора мыши будет изображен символ +), то произойдет не перемещение, а *копирование* выделенного фрагмента.

Недочеты.: при перемещении курсора на участок текста с иным форматированием состояние кнопок быстрого доступа и команд форматирования в меню **Format** не меняется; при загрузке нового файла атрибут `Modified` не сбрасывается; при выполнении команд **New** и **Open** не происходит корректировки форматных настроек в меню и на панели инструментов.

Помимо этих легко выявляемых недочетов, исправленный вариант нашей программы содержит *ошибку*, которую заметить не так просто. Для того чтобы обнаружить эту ошибку, необходимо выделить фрагмент текста, содержащего *более одного вида шрифта* (например, шрифты Microsoft Sans Serif и Arial) и попытаться выполнить одну из команд настройки стиля шрифта (**Bold**, **Italic** или **Underline**) или установить для фрагмента новый вид шрифта с помощью команды **Font**. В подобной ситуации будет возбуждено исключение `NullReferenceException`.

Отмеченные недочеты и ошибки программы будут исправлены в следующем разделе.

23.2. Корректировка состояния кнопок быстрого доступа и команд меню при изменении текущего формата

В класс `Form1` добавьте новый метод `SetEnabled` (листинг 23.2) и определите обработчик события `SelectionChanged` для компонента `textBox1` (листинг 23.3).

В метод `new1_Click` (см. листинг 18.9) после оператора

```
saveFileDialog1.FileName = "";
```

вставьте оператор

```
textBox1_SelectionChanged(this, null);
```

В метод `open1_Click` (см. листинг 23.1) после оператора

```
openFileDialog1.FileName = "";
```

вставьте операторы

```
textBox1.Modified = false;
textBox1_SelectionChanged(this, null);
```

Листинг 23.2. Метод `SetEnabled` формы `Form1`

```
private void SetEnabled(bool value)
{
  bold1.Enabled = bold10.Enabled =
    italic1.Enabled = italic10.Enabled =
    underline1.Enabled = underline10.Enabled =
    font1.Enabled = value;
}
```

Листинг 23.3. Обработчик `textBox1.SelectionChanged`

```
private void textBox1_SelectionChanged(object sender, EventArgs e)
{
  Font f = textBox1.SelectionFont;
```

```
  SetEnabled(f != null);
  if (f != null)
  {
    bold1.Checked = bold10.Checked = f.Bold;
    italic1.Checked = italic10.Checked = f.Italic;
    underline1.Checked = underline10.Checked = f.Underline;
  }
  ToolStripMenuItem mi = leftJustify1;
  switch (textBox1.SelectionAlignment)
  {
    case HorizontalAlignment.Center:
      mi = center1; break;
    case HorizontalAlignment.Right:
      mi = rightJustify1; break;
  }
  if (mi == alignItem) return;
  alignItem.Checked = GetButton(alignItem).Checked = false;
  mi.CheckState = CheckState.Indeterminate;
  GetButton(mi).Checked = true;
  alignItem = mi;
}
```

Результат: исправлены все недочеты, отмеченные в разд. 23.1. В частности, теперь состояние кнопок быстрого доступа и команд меню соответствует формату текущей позиции текста.

Если в тексте выделен фрагмент, то вид команд меню и кнопок быстрого доступа зависит от того, содержит ли выделенный фрагмент участки с различным форматированием. Если во фрагменте используется несколько видов шрифтов, то команды настройки шрифта (**Bold**, **Italic**, **Underline** и **Font**) и связанные с ними кнопки *становятся недоступными* (это позволяет избежать ошибки, отмеченной в разд. 23.1). Если во фрагменте используется один вид шрифта, то указанные команды являются доступными, причем команды настройки стиля шрифта **Bold**, **Italic**, **Underline** и связанные с ними кнопки выделяются только в случае, если *весь* выделенный фрагмент имеет соответствующий стиль шрифта (например, кнопка **Bold** будет выделена только в случае, если для всего фрагмента установлен полужирный стиль).

Аналогичным образом ведут себя команды, связанные с выравниванием текста: если для всего фрагмента установлен одинаковый способ выравнивания, то выделяется команда и кнопка, соответствующие этому способу выравнивания; если же выделенный фрагмент содержит абзацы с *различными* вариантами выравнивания, то выделяется команда **Left justify** и связанная с ней кнопка.

23.3. Настройка свойств абзаца

Добавьте к проекту новую форму `Form2` и разместите в ней компонент-контейнер типа `GroupBox` (он получит имя `groupBox1`). В компоненте `groupBox1` разместите три метки (`label1`, `label2` и `label3`) и три компонента типа `NumericUpDown` (`numericUpDown1`, `numericUpDown2` и `numericUpDown3`). Кроме того, разместите в форме `Form1` еще одну метку (`label4`), выпадающий список (`comboBox1`), флажок (`checkBox1`) и две кнопки (`button1` и `button2`).

Настройте свойства формы `Form2` и ее компонентов (листинг 23.4) и расположите компоненты в соответствии с рис. 23.1. Напомним, что настройка `Modifiers = Internal` позволяет обратиться к компоненту из другой формы этого же проекта.

Дополнительно определите свойство `Items` компонента `comboBox1`. Для этого свойства предусмотрено специальное диалоговое окно (см. рис. 17.2), вызываемое из окна **Properties** нажатием кнопки [...]. В нашем случае в это окно надо ввести три варианта выравнивания, набирая каждый вариант на отдельной строке:

```
По левому краю
По правому краю
По центру
```

Определите обработчик события `CheckedChanged` для компонента `checkBox1` (листинг 23.5).

В описание класса `Form1` добавьте новое поле:

```
private Form2 form2 = new Form2();
```

В конструктор класса `Form1` добавьте оператор

```
AddOwnedForm(form2);
```

Дополните выпадающее меню, связанное с пунктом **Format** *(см. разд. 19.1—19.4)*, добавив в него пункт с текстом **`&Paragraph...`**. Положите свойство `Name` добавленного пункта меню равным **`paragraph1`** и определите для этого пункта меню обработчик события `Click` (листинг 23.6).

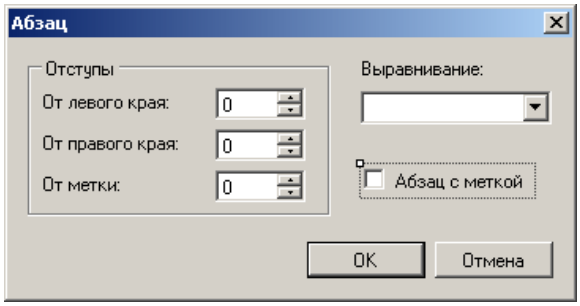

Рис. 23.1. Вид формы Form2 для проекта TXTEDIT6

Листинг 23.4. Настройка свойств

```
Form2: Text = Абзац, MaximizeBox = False,
  MinimizeBox = False,
  FormBorderStyle = FixedDialog,
  StartPosition = CenterScreen,
  ShowInTaskbar = False, AcceptButton = button1,
  CancelButton = button2
groupBox1: Text = Отступы
label1: Text = От левого края:
label2: Text = От правого края:
label3: Text = От метки:, Enabled = False
label4: Text = Выравнивание:
numericUpDown1: Modifiers = Internal
numericUpDown2: Modifiers = Internal
numericUpDown3: Modifiers = Internal,
  Enabled = False
comboBox1: DropDownStyle = DropDownList,
  Modifiers = Internal
checkBox1: Text = Абзац с меткой,
  Modifiers = Internal
button1: Text = OK, DialogResult = OK
button2: Text = Отмена
```

Листинг 23.5. Обработчик `checkBox1.Click` (форма `Form2`)

```
private void checkBox1_Click(object sender, EventArgs e)
{
  label3.Enabled = numericUpDown3.Enabled = checkBox1.Checked;
  numericUpDown3.Value = numericUpDown3.Enabled ? 10 : 0;
}
```

Листинг 23.6. Обработчик `paragraph1.Click` (форма `Form1`)

```
private void paragraph1_Click(object sender, EventArgs e)
{
  form2.checkBox1.Checked = textBox1.SelectionBullet;
  form2.comboBox1.SelectedIndex = (int)textBox1.SelectionAlignment;
  form2.numericUpDown1.Value = textBox1.SelectionIndent;
  form2.numericUpDown2.Value = textBox1.SelectionRightIndent;
  form2.numericUpDown3.Value = textBox1.BulletIndent;
  if (form2.ShowDialog() == DialogResult.OK)
  {
    textBox1.SelectionIndent = (int)form2.numericUpDown1.Value;
    textBox1.SelectionRightIndent = (int)form2.numericUpDown2.Value;
    textBox1.BulletIndent = (int)form2.numericUpDown3.Value;
    textBox1.SelectionBullet = form2.checkBox1.Checked;
    textBox1.SelectionAlignment = (HorizontalAlignment)form2.comboBox1.SelectedIndex;
    textBox1_SelectionChanged(this, null);
  }
}
```

Результат: новая команда меню **Format | Paragraph...** позволяет настраивать свойства текущего абзаца или группы выделенных абзацев. Ее выполнение приводит к появлению диалогового окна **Абзац**, в котором можно указать величину отступов абзаца от левого и правого краев области редактирования (в пикселах) и способ выравнивания. Кроме того, установив флажок **Абзац с меткой**, можно снабдить абзац *меткой* (маркером) в виде черной точки (•); в этом случае можно указать величину отступа от метки до текста.

23.4. Отображение текущей строки и столбца

Действуя таким же образом, как в разд. 22.1, добавьте на статусную панель `statusStrip1` еще одну (пятую) метку с именем **`position1`** и настройте свойства добавленной метки (листинг 23.7).

Зацепите мышью добавленную метку `position1` и переместите ее левее метки `hint1` (с текстом **Ready**). В результате указанные метки поменяются местами (рис. 23.2).

В *начало* метода `textBox1_SelectionChanged`, приведенного в листинге 23.3, добавьте новый фрагмент (листинг 23.8).

Рис. 23.2. Вид нижней части формы Form1 для проекта TXTEDIT6

Листинг 23.6. Настройка свойств

```
position1: Text = 1 : 1, AutoSize = False,
  BorderSides = All, BorderStyle = Sunken,
  Size.Width = 60
```

Листинг 23.7. Добавление в начало метода `textBox1_SelectionChanged`

```
int x = textBox1.SelectionStart,
  y = textBox1.GetLineFromCharIndex(x),
  x0 = textBox1.GetFirstCharIndexFromtLine(y);
position1.Text = string.Format("{0} : {1}", y + 1, x - x0 + 1);
```

Результат: на статусной панели отображается текущая позиция клавиатурного курсора *(каретки)* в формате *номер строки* **:** *номер символа в строке* (строки и символы нумеруются от 1). Если редактируемый текст содержит выделенный фрагмент, то указывается позиция начала этого фрагмента.

23.5. Загрузка и сохранение текста без форматных настроек

Для компонентов `saveFileDialog1` и `openFileDialog1` измените значения свойства `Filter` на **`RTF files|*.rtf|Text files|*.txt|All files|*.*`**. В результате с указанными диалоговыми окнами будут связаны *три* фильтра: для RTF-файлов (с расширением rtf), обычных текстовых файлов (с расширением txt) и произвольных файлов, имеющих любое допустимое имя и расширение.

В класс `Form1` добавьте новый метод `GetFileType` (листинг 23.9).

Измените методы `SaveToFile` и `open1_Click` (листинг 23.10).

Листинг 23.9. Метод `GetFileType` формы `Form1`

```
private RichTextBoxStreamType GetFileType(string path)
{
  string s = Path.GetExtension(path).ToUpper();
  return s == ".RTF" ? RichTextBoxStreamType.RichText : RichTextBoxStreamType.PlainText;
}
```

Листинг 23.10. Новый вариант методов `SaveToFile` и `open1_Click`

```
private void SaveToFile(string path)
{
  textBox1.SaveFile(path, GetFileType(path));
  textBox1.Modified = false;
}
private void open1_Click(object sender, EventArgs e)
{
  if (TextSaved())
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
      string path = openFileDialog1.FileName;
      textBox1.LoadFile(path, GetFileType(path));
      Text = "Text Editor - " + Path.GetFileName(path);
      saveFileDialog1.FileName = path;
      openFileDialog1.FileName = "";
    }
}
```

Результат: при сохранении документа в файле с любым расширением, отличным от rtf, в файл записывается только текст (без форматных настроек), причем в *ANSI-кодировке*, то есть в кодировке, используемой системой Windows по умолчанию. Однако в самом редакторе форматные настройки сохраняются, и в дальнейшем текст вместе с форматными настройками можно сохранить в файле с расширением rtf. Загрузить в редактор теперь можно как файлы в формате RTF (с расширением rtf), так и обычные текстовые файлы. Для того чтобы ускорить выбор файлов с расширениями rtf и txt, а также иметь возможность выбора файлов с любым расширением, в окнах открытия и сохранения файлов предусмотрены соответствующие *фильтры* (**RTF files**, **Text files**, **All files**), перечисленные в выпадающем списке **Тип файла**.

Приведем изображение работающей программы (рис. 23.3). Клавиатурный курсор располагается на первой, отцентрированной, строке.

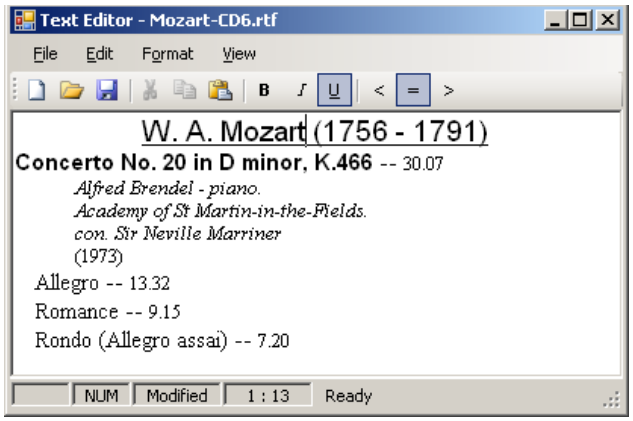

Рис. 23.3. Вид работающего приложения TXTEDIT6