Глава 17. Дополнительные графические возможности: проект PNGEDIT4

17.1. Рисование квадратов и окружностей

В качестве заготовки для проекта PNGEDIT4 следует использовать ранее разработанный проект PNGEDIT3 (см. гл. 16).

В класс `Form1` добавьте новое поле:

```
private bool equalSize;
```

Измените методы `PtToRect` и `pictureBox1_MouseMove` (листинг 17.1).

Листинг 17.1. Новый вариант методов `PtToRect` и `pictureBox1_MouseMove`

```
private Rectangle PtToRect(Point p1, Point p2)
{
  if (equalSize)
  {
    int dx = p2.X - p1.X, dy = p2.Y - p1.Y;
    if (Math.Abs(dx) > Math.Abs(dy))
      p2.X = p1.X + Math.Sign(dx) * Math.Abs(dy);
    else
      p2.Y = p1.Y + Math.Sign(dy) * Math.Abs(dx);
  }
  int x = Math.Min(p1.X, p2.X),
    y = Math.Min(p1.Y, p2.Y),
    w = Math.Abs(p2.X - p1.X),
    h = Math.Abs(p2.Y - p1.Y);
  return new Rectangle(x, y, w, h);
}
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
  label1.Text = string.Format("X,Y: {0},{1}", e.X, e.Y);
  if (startPt == nullPt)
    return;
  if (e.Button == MouseButtons.Left)
    switch (mode)
    {
      case 0:
        Graphics g = Graphics.FromImage(pictureBox1.Image);
        g.DrawLine(pen, startPt, e.Location);
        g.Dispose();
        startPt = e.Location;
        pictureBox1.Invalidate();
        break;
      case 1:
      case 2:
        ReversibleDraw();
        movePt = e.Location;
        equalSize = Control.ModifierKeys == Keys.Control;
        ReversibleDraw();
        break;
    }
}
```

Результат: теперь в режиме **Figure** можно рисовать фигуры с равными измерениями (квадраты и окружности). Для этого в процессе рисования следует держать нажатой клавишу <Ctrl>. Допускается нажимать и отпускать эту клавишу после начала перемещения мыши; при этом инверсный контур будет автоматически корректироваться, изображая при нажатой <Ctrl> квадратную, а в противном случае — прямоугольную рамку.

17.2. Отмена предыдущей операции

В класс `Form1` добавьте новое поле:

```
private Bitmap oldImage;
```

В конструктор класса `Form1` добавьте оператор:

```
oldImage = new Bitmap(pictureBox1.Image);
```

В класс `Form1` добавьте новый метод `UpdateOldImage` (листинг 17.2). Вставьте вызов добавленного метода в обработчики `button1_Click`, `button2_Click`, `button4_Click` и `pictureBox1_MouseDown` (листинг 17.3).

Установите свойство `KeyPreview` формы `Form1` равным **`True`** и определите обработчик события `KeyDown` для формы `Form1` (листинг 17.4).

Листинг 17.2. Метод `UpdateOldImage` формы `Form1`

```
private void UpdateOldImage()
{
```

```
  oldImage.Dispose();
  oldImage = new Bitmap(pictureBox1.Image);
}
```

Листинг 17.3. Новый вариант методов button1_Click, button2_Click, button4_Click и pictureBox1_MouseDown

```
private void button1_Click(object sender, EventArgs e)
{
  form2.ActiveControl = form2.numericUpDown1;
  if (form2.ShowDialog() == DialogResult.OK)
  {
    saveFileDialog1.FileName = "";
    Text = "Image Editor";
    UpdateOldImage();
  }
}
private void button2_Click(object sender, EventArgs e)
{
  startPt = nullPt;
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
    string s = openFileDialog1.FileName;
    try
    {
      Image im = new Bitmap(s);
      Graphics g = Graphics.FromImage(im);
      g.Dispose();
      if (pictureBox1.Image != null)
        pictureBox1.Image.Dispose();
      pictureBox1.Image = im;
      UpdateOldImage();
    }
    catch
    {
      MessageBox.Show("File " + s + " has a wrong format.","Error");
      return;
    }
    Text = "Image Editor - " + s;
    saveFileDialog1.FileName = Path.ChangeExtension(s, "png");
    openFileDialog1.FileName = "";
  }
}
private void button4_Click(object sender, EventArgs e)
{
  UpdateOldImage();
  using (Graphics g = Graphics.FromImage(pictureBox1.Image))
    g.Clear(brush.Color);
  pictureBox1.Invalidate();
}
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
  movePt = startPt = e.Location;
  UpdateOldImage();
}
```

Листинг 17.4. Обработчик Form1.KeyDown

```
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
  if (e.KeyCode == Keys.Escape)
  {
    pictureBox1.Image.Dispose();
    pictureBox1.Image = new Bitmap(oldImage);
  }
}
```

Результат: теперь любую графическую операцию (в том числе и закрашивание изображения фоновым цветом, выполняемую по нажатию кнопки **Clear**) можно отменить сразу после ее выполнения, нажав клавишу <Esc>.

17.3. Задание цветов с помощью пипетки

Дополните метод pictureBox1_MouseDown (листинг 17.5).

Листинг 17.5. Новый вариант метода pictureBox1_MouseDown

```
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
{
  movePt = startPt = e.Location;
```

```
  UpdateOldImage();
  if (Control.ModifierKeys == Keys.Alt)
  {
    Color c = (pictureBox1.Image as Bitmap).GetPixel(e.X, e.Y);
    if (e.Button == MouseButtons.Left)
      label2.BackColor = c;
    else
      label4.BackColor = c;
  }
}
```

Результат: новый цвет линии и фона теперь можно получить непосредственно из текущего изображения, "втянув его пипеткой", а именно — щелкнув соответственно левой или правой кнопкой мыши на нужном месте изображения при нажатой клавише <Alt>. Если нажата левая кнопка мыши, то можно, не отпуская ее, сразу приступить к рисованию новым цветом.

17.4. Четвертый режим рисования: добавление в рисунок текста

В компоненте `groupBox1` разместите еще одну радиокнопку (`radioButton4`). Добавьте в форму `Form1` метку `label6`, поле ввода `textBox1`, кнопку `button5` и невизуальный компонент типа `FontDialog` (он получит имя `fontDialog1` и будет размещен ниже формы, в области невизуальных компонентов). Настройте свойства добавленных компонентов (листинг 17.6; рис. 17.1).

Свяжите обработчик `radioButton1_CheckedChanged` (см. листинг 15.4) с событием `CheckedChanged` добавленной радиокнопки `radioButton4`.

Добавьте в класс `Form1` новое поле:

```
private Font font;
```

В конструктор класса `Form1` добавьте оператор

```
font = Font.Clone() as Font;
```

В метод `pictureBox1_MouseDown` (листинг 17.5) добавьте новый фрагмент (листинг 17.7) и определите обработчики события `Enter` для поля ввода `textBox1` и события `Click` для кнопки `button5` (листинг. 17.8).

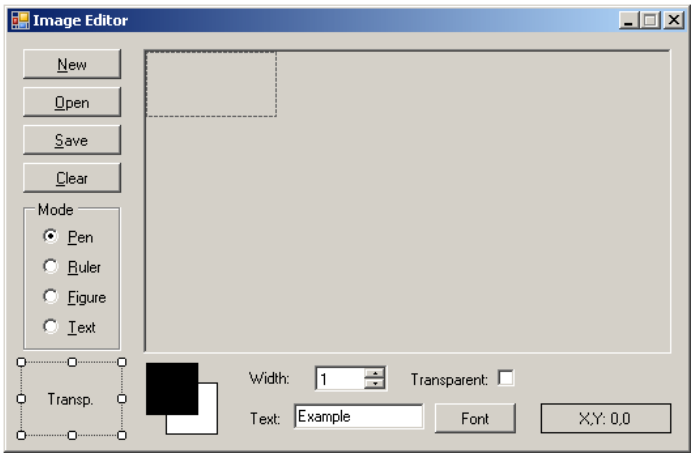

Рис. 17.1. Вид формы Form1 для проекта PNGEDIT4

Листинг 17.6. Настройка свойств

```
radioButton4: Text = &Text
label6: Text = Text:
textBox1: Text = Example
button5: Text = Font
fontDialog1: MinSize = 8, MaxSize = 28
```

Листинг 17.7. Добавление к методу `pictureBox1_MouseDown`

```
else
  if (mode == 3)
  {
    Graphics g = Graphics.FromImage(pictureBox1.Image);
    using (SolidBrush b = new SolidBrush(pen.Color))
      g.DrawString(textBox1.Text, font, b, e.Location);
    g.Dispose();
    pictureBox1.Invalidate();
  }
```

Листинг 17.8. Обработчики `textBox1.Enter` и `button5.Click`

```
private void textBox1_Enter(object sender, EventArgs e)
{
```

```
  radioButton4.Checked = true;
}
private void button5_Click(object sender, EventArgs e)
{
  fontDialog1.Font = font;
  if (fontDialog1.ShowDialog() == DialogResult.OK)
  {
    Font f = font;
    textBox1.Font = font = fontDialog1.Font;
    f.Dispose();
  }
}
```

Результат: новый режим **Text** позволяет вставлять в изображение текст, содержащийся в поле ввода `textBox1`. Текст вставляется при щелчке мышью на изображении; место щелчка определяет начальную позицию текста, точнее, его левый верхний угол. Текст выводится текущим цветом пера; при выводе используется шрифт, установленный для формы (Microsoft Sans Serif, 8,25 пунктов). Операция вставки текста, как и прочие графические операции, может быть отменена сразу после выполнения нажатием клавиши <Esc>. Режим **Text** автоматически устанавливается при активизации поля ввода `textBox1`, т. е. при получении им фокуса; это действие обеспечивает обработчик `textBox1_Enter` (см. листинг 17.8).

Нажатие кнопки **Font** приводит к появлению диалогового окна **Шрифт**, позволяющего выбирать нужный шрифт и настраивать такие характеристики шрифта, как начертание, размер и набор символов. Благодаря установленным значениям свойств компонента `fontDialog1` (см. листинг 17.6), размеры шрифта в диалоговом окне можно устанавливать в пределах от 8 до 28 пунктов. При открытии диалогового окна в нем отображается текущая настройка шрифта. При выборе нового шрифта и закрытии диалогового окна кнопкой **OK** текст в поле ввода `textBox1` отображается выбранным шрифтом.

17.5. Настройка стиля изображения линии

Разместите в форме `Form1` над меткой `label1` *выпадающий список* — компонент типа `ComboBox` (он получит имя `comboBox1`) и настройте его свойства (листинг 17.9; при определении свойства `Items` используется специальное диалоговое окно **String Collection Editor**, в нашем случае в это окно надо ввести указанные 5 чисел, набирая каждое число в *отдельной* строке — см. рис. 17.2).

В конструктор класса `Form1` добавьте оператор

```
comboBox1.SelectedIndex = 0;
```

и определите обработчики событий `DrawItem` и `SelectedIndexChanged` для компонента `comboBox1` (листинг. 17.10).

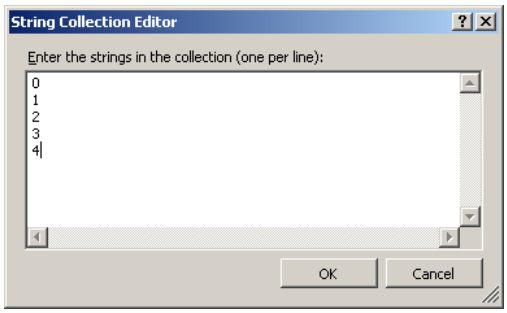

Рис. 17.2. Вид окна настройки свойства `Items`

Листинг 17.9. Настройка свойств

```
comboBox1: DrawMode = OwnerDrawFixed,
  DropDownStyle = DropDownList, Items: 0 1 2 3 4
```

Листинг 17.10. Обработчики `comboBox1.DrawItem` и `comboBox1.SelectedIndexChanged`

```
private void comboBox1_DrawItem(object sender, DrawItemEventArgs e)
{
  e.DrawBackground();
  // - заливка фоновым цветом области,
  // связанной с рисуемым элементом списка
  using (Pen p = new Pen(e.ForeColor, 2))
  {
    // рисование образцов линий в элементах
    // выпадающего списка:
    p.DashStyle = (DashStyle)e.Index;
    int y = (e.Bounds.Top + e.Bounds.Bottom) / 2;
    e.Graphics.DrawLine(p, e.Bounds.Left, y, e.Bounds.Right, y);
  }
  e.DrawFocusRectangle();
  // - дополнительное выделение
  // текущего элемента списка
}
```

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  pen.DashStyle = (DashStyle)comboBox1.SelectedIndex;
  label5.Invalidate();
}
```

Результат: форма `Form1` теперь содержит выпадающий список, который позволяет выбирать все возможные стили рисования линий. Варианты стилей изображаются в выпадающем списке в графическом виде. Отметим, что в GDI+ штриховые линии могут иметь любую ширину (тогда как в прежней библиотеке GDI штриховые линии могли иметь ширину только в 1 пиксел).

Примечание

В режиме **Pen** линии всегда изображаются сплошными; это связано с особенностями реализации данного режима в нашей программе. Таким образом, штриховые линии можно использовать только в режимах **Ruler** и **Figure** (то есть при рисовании прямых линий и контуров фигур).

Приведем изображение работающей программы с развернутым списком `comboBox1` (рис. 17.3).

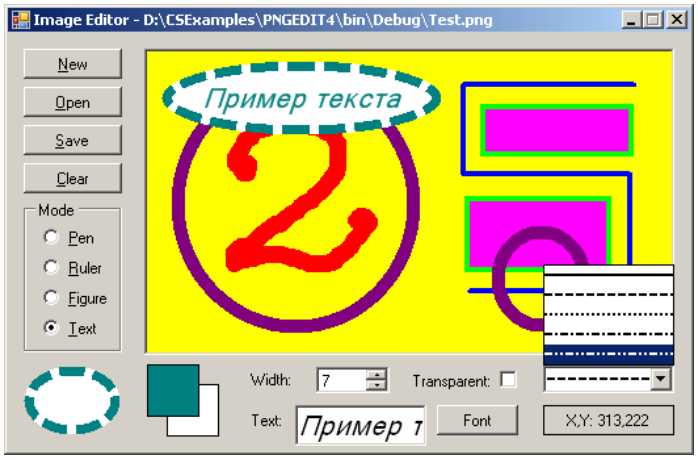

Рис. 17.3. Вид работающего приложения PNGEDIT4

Глава 19. Дополнительные возможности меню, настройка цвета и шрифта: проект TXTEDIT2

19.1. Установка начертания символов (команды меню — флажки)

В качестве заготовки для проекта TXTEDIT2 следует использовать ранее разработанный проект TXTEDIT1 (см. гл. 18). Скопируйте проект TXTEDIT1 в новый каталог TXTEDIT2 и выполните действия, необходимые для переименования проекта (см. разд. 1.1).

Действуя так же, как разд. 18.1 при создании пункта меню **File** и связанных с ним пунктов меню второго уровня, создайте в компоненте `menuStrip1` новый пункт меню первого уровня с текстом **`F&ormat`** и с помощью окна **Properties** измените имя этого пункта (т. е. его свойство `Name`) на **`format1`**. В выпадающем меню, связанном с пунктом **Format**, создайте пункты с текстом **`&Bold`**, **`&Italic`**, **`&Underline`**. Настройте свойства добавленных пунктов меню (листинг 19.1; обратите внимание на свойство `Font`, для которого надо изменить одно из его свойств: `Bold`, `Italic` или `Underline`). Вид полученного меню приведен на рис. 19.1.

Определите обработчик события `Click` для пункта меню `bold1` (листинг 19.2), после чего свяжите созданный обработчик с событием `Click` пунктов меню `italic1` и `underline1` (связывание имеющихся обработчиков с событиями пунктов меню выполняется так же, как и аналогичное связывание для обычных компонентов — см. разд. 6.1).

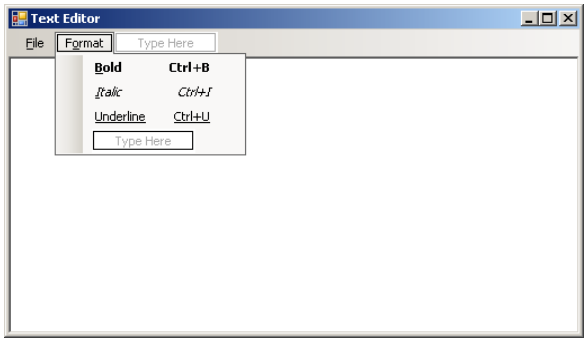

Рис. 19.1. Вид формы Form1 для проекта TXTEDIT2 на начальном этапе разработки

Листинг 19.1. Настройка свойств

```
пункт меню Bold (группа Format): Name = bold1,
  ShortcutKeys = Ctrl+B, Font.Bold = True
пункт меню Italic (группа Format): Name = italic1,
  ShortcutKeys = Ctrl+I, Font.Italic = True
пункт меню Underline (группа Format):
  Name = underline1, ShortcutKeys = Ctrl+U,
  Font.Underline = True
```

Листинг 19.2. Обработчик `bold1.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();
}
```

Результат: при выполнении добавленных в меню команд устанавливается соответствующий стиль *(начертание)* символов в редакторе: **Bold** — полужирный, **Italic** — курсивный, **Underline** — подчеркнутый. При вызове меню **Format** около команд с установленными стилями изображаются "галочки" (как для установленных флажков). При повторном выполнении команды соответствующий стиль отменяется, и галочка исчезает. Заметим, что названия указанных команд выводятся в меню тем стилем, который они настраивают (например, название команды **Bold** выводится полужирным стилем). Подобное наглядное отображение пунктов меню имеет единственный недостаток: в названии команды **Underline** нельзя определить, какая буква связана с "горячей" клавишей (поскольку подчеркнутыми являются все буквы).

19.2. Установка выравнивания текста (команды меню — переключатели)

Дополните выпадающее меню, связанное с пунктом **Format** (см. разд. 19.1). Вначале добавьте в него пункт с текстом **–** (дефис); этот пункт будет преобразован в горизонтальную разделительную линию. Затем добавьте пункты меню с текстом **&Left justify**, **C&enter** и **&Right justify** и настройте свойства этих пунктов (листинг 19.3).

В класс `Form1` добавьте новое поле:

```
private ToolStripMenuItem alignItem;
```

В конструктор класса `Form1` добавьте новые операторы (листинг 19.4).

Определите обработчик события `Click` для пункта меню `leftJustify1` (листинг 19.5), после чего свяжите созданный обработчик с событием `Click` пунктов меню `center1` и `rightJustify1`.

Листинг 19.3. Настройка свойств

```
пункт меню Left justify (группа Format):
  Name = leftJustify1, ShortcutKeys = Ctrl+L,
  CheckState = Indeterminate
пункт меню Center (группа Format): Name = center1,
  ShortcutKeys = Ctrl+E
пункт меню Right justify (группа Format):
  Name = rightJustify1, ShortcutKeys = Ctrl+R
```

Листинг 19.4. Добавление к конструктору формы **Form1**

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

Листинг 19.5. Обработчик `leftJustify1.Click`

```
private void leftJustify1_Click(object sender, EventArgs e)
{
  ToolStripMenuItem mi = sender as ToolStripMenuItem;
  if (mi.Checked) return;
  alignItem.Checked = false;
  alignItem = mi;
  mi.CheckState = CheckState.Indeterminate;
  textBox1.TextAlign = (HorizontalAlignment)mi.Tag;
}
```

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 метка • (как для выбранной радиокнопки). Таким образом, добавленные в данном разделе команды меню ведут себя как *группа переключателей*.

19.3. Установка цвета символов и фона (команды меню третьего уровня и окно диалога *Цвет*)

Добавьте в форму `Form1` невизуальный компонент типа `ColorDialog` (он получит имя `colorDialog1` и будет размещен в области невизуальных компонентов под изображением формы).

Дополните выпадающее меню, связанное с пунктом **Format** (см. разд. 19.1 и 19.2), добавив в него еще одну горизонтальную разделительную линию и пункт с текстом **`&Colors`**. Затем перейдите в заготовку меню третьего уровня, связанную с пунктом **Colors**, и добавьте в нее пункты меню с текстом **`&Font color...`** и **`&Background color...`**. Настройте свойство `Name` добавленных пунктов меню (листинг 19.6) и определите обработчики события `Click` для пунктов меню `fontColor1` и `backgroundColor1` (листинг 19.7). Вид полученного меню группы **Format** приведен на рис. 19.2 (последний пункт этого меню с именем **Font...** будет добавлен в следующем разделе).

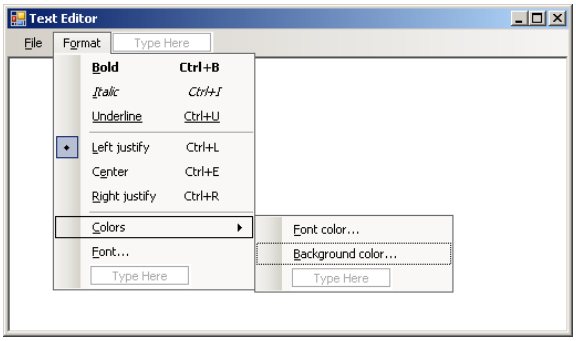

Рис. 19.2. Окончательный вид формы Form1 для проекта TXTEDIT2

Листинг 19.6. Настройка свойств

```
пункт меню Colors (группа Format): Name = colors1
пункт меню Font color… (группа Colors):
  Name = fontColor1
пункт меню Background color… (группа Colors):
  Name = background1
```

Листинг 19.7. Обработчики `fontColor1.Click` и `backgroundColor1.Click`

```
private void fontColor1_Click(object sender, EventArgs e)
{
  colorDialog1.Color = textBox1.ForeColor;
  if (colorDialog1.ShowDialog() == DialogResult.OK)
    textBox1.ForeColor = colorDialog1.Color;
}
private void backgroundColor1_Click(object sender, EventArgs e)
{
  colorDialog1.Color = textBox1.BackColor;
  if (colorDialog1.ShowDialog() == DialogResult.OK)
    textBox1.BackColor = 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 обводится рамкой текущий цвет. Для установки нового цвета надо выбрать его и нажать кнопку **OK**.

19.4. Установка свойств шрифта с помощью окна диалога *Шрифт*

Добавьте в форму `Form1` невизуальный компонент типа `FontDialog` (он получит имя `fontDialog1` и будет размещен в области невизуальных компонентов под изображением формы).

Дополните выпадающее меню, связанное с пунктом **Format** (см. разд. 19.1–19.3), добавив в него пункт меню с текстом **`&Font`**… (см. рис. 19.2). Положите свойство `Name` добавленного пункта меню равным **`font1`** и определите для данного пункта меню обработчик события `Click` (листинг 19.8).

Листинг 19.8. Обработчик `font1.Click`

```
private void font1_Click(object sender, EventArgs e)
{
  fontDialog1.Font = textBox1.Font;
  if (fontDialog1.ShowDialog() == DialogResult.OK)
  {
    Font f = textBox1.Font;
    textBox1.Font = fontDialog1.Font;
    f.Dispose();
    bold1.Checked = fontDialog1.Font.Bold;
    italic1.Checked = fontDialog1.Font.Italic;
```

```
    underline1.Checked = fontDialog1.Font.Underline;
  }
}
```

Результат: при выполнении команды **Font…** вызывается диалоговое окно **Шрифт**, позволяющее изменить шрифт в редакторе. При отображении диалогового окна в нем указываются текущие настройки шрифта. В случае изменения стиля шрифта дополнительно производится корректировка флажков для пунктов меню **Bold**, **Italic** и **Underline**.

Ошибка: если открыть диалоговое окно **Шрифт** и, *не изменяя никаких свойств шрифта*, закрыть окно, нажав кнопку **OK** или клавишу <Enter>, то при последующем открытии окна **Шрифт** произойдет ошибка времени выполнения. Эта ошибка возникает "внутри" метода `ShowDialog` и связана со взаимодействием свойств `Font` объектов `fontDialog1` и `textBox1`, в частности, с удалением старого варианта шрифта для компонента `textBox1`. К последнему выводу можно прийти, заметив, что если закомментировать оператор `f.Dispose()`, то указанная ошибка не возникает.

Итак, для исправления ошибки можно было бы не вызывать метод `Dispose`. Но это плохой вариант исправления, так как он будет приводить к излишнему расходованию ресурсов системы. Разумеется, при отсутствии иных альтернатив таким вариантом можно воспользоваться, однако мы можем выбрать лучший вариант.

Исправление: добавьте в метод `font1_Click` еще один условный оператор (листинг 19.9).

Листинг 19.9. Новый вариант метода `font1_Click`

```
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();
      bold1.Checked = fontDialog1.Font.Bold;
      italic1.Checked = fontDialog1.Font.Italic;
      underline1.Checked = fontDialog1.Font.Underline;
    }
}
```

Результат: теперь в описанной выше ситуации ошибки времени выполнения не происходит.

Глава 20. Команды редактирования, контекстное меню: проект TXTEDIT3

20.1. Команды редактирования

В качестве заготовки для проекта TXTEDIT3 следует использовать ранее разработанный проект TXTEDIT2 (см. гл. 19).

Перейдите в конструктор меню (см. разд. 18.1) и вставьте новый пункт меню первого уровня *перед* уже имеющимся пунктом **Format**. Для этого выделите пункт **Format**, вызовите его контекстное меню (нажав правую кнопку мыши) и выберите в нем команду **Insert | MenuItem**. Новый пункт меню сразу получит имя `toolStripMenuItem`*N* (где *N* — некоторое число) и заголовок, совпадающий с этим именем. Заголовок нового пункта меню замените на **`&Edit`** (это можно сделать в самом конструкторе меню), а его имя, то есть свойство `Name`, замените на **`edit1`** с помощью окна **Properties**.

В выпадающем меню, связанном с новым пунктом **Edit**, создайте пункты меню с текстом **`&Undo`**, дефисом **`-`** (этот пункт будет преобразован в горизонтальную линию), **`Cu&t`**, **`&Copy`**, **`&Paste`**, **`&Delete`**, еще одним дефисом **`-`**, **`&Select All`**. Настройте свойства добавленных пунктов меню (листинг 20.1); в результате меню **Edit** примет вид, приведенный на рис. 20.1.

Определите для пунктов меню **Edit** обработчики события `Click` (листинг 20.2).

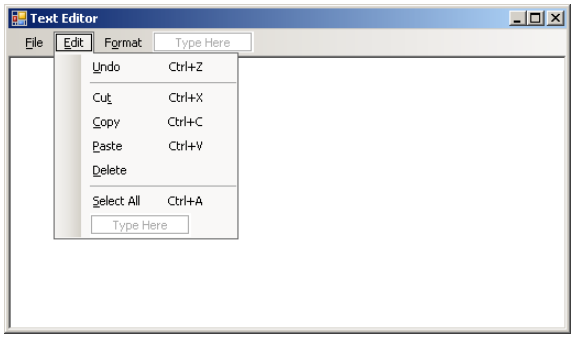

Рис. 20.1. Вид формы Form1 для проекта TXTEDIT3 с развернутым меню **Edit**

Листинг 20.1. Настройка свойств

```
пункт меню Undo (группа Edit): Name = undo1,
  ShortcutKeys = Ctrl+Z
пункт меню Cut (группа Edit): Name = cut1,
  ShortcutKeys = Ctrl+X
```

```
пункт меню Copy (группа Edit): Name = copy1,
  ShortcutKeys = Ctrl+C
пункт меню Paste (группа Edit): Name = paste1,
  ShortcutKeys = Ctrl+V
пункт меню Delete (группа Edit): Name = delete1
пункт меню Select All (группа Edit):
  Name = selectAll1, ShortcutKeys = Ctrl+A
```

Листинг 20.2. Обработчики `undo1.Click`, `cut1.Click`, `copy1.Click`, `paste1.Click`, `delete1.Click`, `selectAll1.Click`

```
private void undo1_Click(object sender, EventArgs e)
{
  textBox1.Undo();
}
private void cut1_Click(object sender, EventArgs e)
{
  textBox1.Cut();
}
private void copy1_Click(object sender, EventArgs e)
{
  textBox1.Copy();
}
private void paste1_Click(object sender, EventArgs e)
{
  textBox1.Paste();
}
private void delete1_Click(object sender, EventArgs e)
{
  textBox1.SelectedText = "";
}
private void selectAll1_Click(object sender, EventArgs e)
{
  textBox1.SelectAll();
}
```

Результат: в меню определены стандартные команды редактирования: отмена последнего действия **Undo**, вырезание **Cut** и копирование **Copy** выделенного фрагмента в буфер, вставка **Paste** фрагмента из буфера, удаление выделенного фрагмента **Delete**, выделение всего текста **Select All**.

20.2. Выделение недоступных команд редактирования. Работа с буфером обмена

Определите обработчик события `Click` для пункта меню `edit1` (листинг 20.3).

Листинг 20.3. Обработчик `edit1.Click`

```
private void edit1_Click(object sender, EventArgs e)
{
  undo1.Enabled = textBox1.CanUndo;
  cut1.Enabled = copy1.Enabled = delete1.Enabled =
    textBox1.SelectionLength > 0;
  paste1.Enabled =
    Clipboard.GetDataObject().GetDataPresent(typeof(string));
  selectAll1.Enabled = textBox1.Text != "";
}
```

Результат: при вызове подменю группы **Edit** недоступные пункты меню выделяются серым цветом. Пункт **Undo** доступен, если ранее было выполнено действие, которое можно отменить. Пункты **Cut**, **Copy** и **Delete** доступны, если текст содержит выделенный фрагмент. Пункт **Paste** доступен, если буфер обмена содержит данные в текстовом формате. Пункт **Select All** доступен, если в редакторе содержится непустой текст.

Недочет: обработчик `edit1_Click` выполняется уже *после* разворачивания подменю команды **Edit**, поэтому выделение недоступных пунктов меню происходит на глазах пользователя, что выглядит неряшливо.

Исправление: находясь в окне **Properties** в режиме **Events**, выполните для компонента **edit1** следующие действия:

- из строки **Click** удалите текст **`edit1_Click`**;
- в строку **DropDownOpening** добавьте текст **`edit1_Click`** (для этого проще всего воспользоваться выпадающим списком доступных обработчиков).

Результат: теперь указанные в обработчике действия выполняются после щелчка на пункте **Edit**, но *перед* разворачиванием подменю, поскольку именно в этот момент происходит событие `DropDownOpening`, с которым мы связали обработчик.

20.3. Создание контекстного меню

Добавьте в форму `Form1` компонент типа `ContextMenuStrip` (он получит имя `contextMenuStrip1` и будет размещен под изображением формы, в области невизуальных компонентов). Данный компонент позволяет использовать в приложении

контекстные меню. Свяжите компонент `contextMenuStrip1` с областью редактирования (компонентом `textBox1`), положив свойство `ContextMenuStrip` компонента `textBox1` равным **`contextMenuStrip1`.**

Для определения команд контекстного меню, как и в случае обычного меню, предназначен *конструктор меню*. Однако, в отличие от обычного меню, конструктор для контекстного меню отображается в форме только при выделении компонента, связанного с контекстным меню (в нашем случае — компонента `contextMenuStrip1`). Горизонтальное меню первого уровня для контекстного меню создавать нельзя; не рекомендуется также создавать в контекстном меню вложенные выпадающие меню и связывать с командами выпадающего меню клавиши-ускорители.

Действуя так же, как в разд. 18.1 при создании пункта меню **File** и связанных с ним пунктов меню второго уровня, создайте в контекстном меню `contextMenuStrip1` пункты меню с текстом **`Cu&t`**, **`&Copy`**, **`&Paste`**, дефис **`-`** (который будет преобразован в горизонтальную линию), **`&Font...`** (рис. 20.2). Настройте свойства добавленных пунктов меню (листинг 20.4; обратите внимание на то, что при определении имен для пунктов контекстного меню мы используем не цифру 1, как для пунктов главного меню, а цифру 2). В листинге 20.4 также указано, с какими существующими обработчиками надо связать событие `Click` для каждой из команд контекстного меню.

Кроме того, определите обработчик события `Opening` для компонента `contextMenuStrip1` (листинг 20.5).

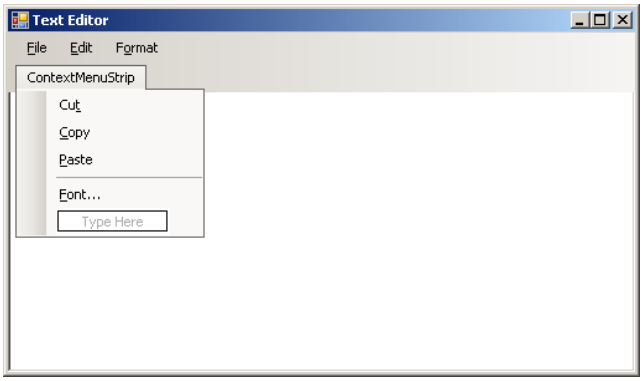

Рис. 20.2. Вид формы Form1 для проекта TXTEDIT3 с конструктором контекстного меню

Листинг 20.4. Настройка свойств

```
пункт Cut (меню contextMenuStrip1): Name = cut2,
  Click = cut1_Click
пункт Copy (меню contextMenuStrip1): Name = copy2,
  Click = copy1_Click
пункт Paste (меню contextMenuStrip1): Name =
  paste2, Click = paste1_Click
пункт Font (меню contextMenuStrip1): Name = font2,
  Click = font1_Click
```

Листинг 20.5. Обработчик `contextMenuStrip1.Opening`

```
private void contextMenuStrip1_Opening(object sender, CancelEventArgs e)
{
  cut2.Enabled = copy2.Enabled = textBox1.SelectionLength > 0;
  paste2.Enabled =
    Clipboard.GetDataObject().GetDataPresent(typeof(string));
}
```

Результат: при нажатии правой кнопки мыши в окне редактора вместо стандартного контекстного меню компонента `TextBox` отображается контекстное меню, определенное в компоненте `contextMenuStrip1`. Недоступные в данный момент команды контекстного меню выделяются серым цветом.