Глава 14. Работа с графическими файлами, рисование тонким пером: проект PNGEDIT1

14.1. Создание, сохранение и загрузка графических файлов

После создания проекта PNGEDIT1 добавьте в форму `Form1` три кнопки (`button1`—`button3`), панель `panel1` и невизуальные компоненты типа `OpenFileDialog` и `SaveFileDialog` (эти компоненты получат имена `openFileDialog1` и `saveFileDialog1`; они будут размещены в области невизуальных компонентов под изображением формы). На панели `panel1` разместите компонент типа `PictureBox` (он получит имя `pictureBox1`). Настройте свойства формы `Form1` и всех добавленных компонентов (листинг 14.1) и расположите компоненты в соответствии с рис. 14.1.

Добавьте к проекту новую форму (она получит имя `Form2`) и разместите в ней две метки (`label1` и `label2`), два компонента типа `NumericUpDown` (`numericUpDown1` и `numericUpDown2`) и две кнопки (`button1` и `button2`). Настройте свойства формы `Form2` и ее компонентов (листинг 14.2) и расположите компоненты в соответствии с рис. 14.2 (компонент `numericUpDown1` должен находиться рядом с меткой `label1`, а компонент `numericUpDown2` — рядом с меткой `label2`).

В начало файла Form1.cs добавьте оператор

```
using System.IO;
```

В описание класса `Form1` добавьте поле

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

В конструктор класса `Form1` добавьте новые операторы (листинг 14.3) и определите обработчики события `Click` для кнопок `button1`–`button3`, размещенных в форме `Form1` (листинг 14.4).

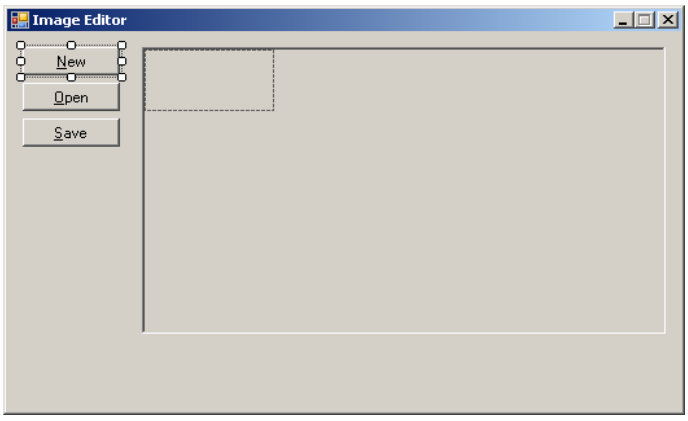

Рис. 14.1. Вид формы Form1 для проекта PNGEDIT1 на начальном этапе разработки

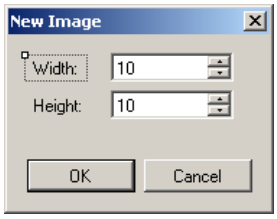

Рис. 14.2. Вид формы Form2 для проекта PNGEDIT1

Листинг 14.1. Настройка свойств формы `Form1` и ее компонентов

```
Form1: Text = Image Editor, MaximizeBox = False,
  FormBorderStyle = FixedSingle,
  StartPosition = CenterScreen
button1: Text = &New
button2: Text = &Open
button3: Text = &Save
panel1: AutoScroll = True, BorderStyle = Fixed3D
pictureBox1: Location = 0; 0, SizeMode= AutoSize
openFileDialog1: DefaultExt = png,
  FileName = пустая строка,
  Filter = Image files (*.bmp, *.jpg, *.png, *.gif)|*.bmp;*.jpg;*.png;*.gif
saveFileDialog1: DefaultExt = png,
  Filter = PNG-files (*.png) |*.png
```

Листинг 14.2. Настройка свойств формы `Form2` и ее компонентов

```
Form2: Text = NewImage, MaximizeBox = False,
  MinimizeBox = False,
  FormBorderStyle = FixedDialog,
```

```
  StartPosition = CenterScreen,
  ShowInTaskbar = False, AcceptButton = button1,
  CancelButton = button2
label1: Text = Width:
label2: Text = Height:
numericUpDown1: Minimum = 10, Maximum = 800,
  Increment = 10, Modifiers = Internal
numericUpDown2: Minimum = 10, Maximum = 600,
  Increment = 10, Modifiers = Internal
button1: Text = OK, DialogResult = OK
button2: Text = Cancel
```

Листинг 14.3. Новый вариант конструктора формы `Form1`

```
public Form1()
{
  InitializeComponent();
  AddOwnedForm(form2);
  openFileDialog1.InitialDirectory = saveFileDialog1.InitialDirectory = Directory.GetCurrentDirectory();
  form2.numericUpDown1.Value = panel1.ClientSize.Width;
  form2.numericUpDown2.Value = panel1.ClientSize.Height;
}
```

Листинг 14.4. Обработчики `button1.Click`, `button2.Click` и `button3.Click` (для кнопок формы `Form1`)

```
private void button1_Click(object sender, EventArgs e)
{
  form2.ActiveControl = form2.numericUpDown1;
  if (form2.ShowDialog() == DialogResult.OK)
  {
    saveFileDialog1.FileName = "";
    Text = "Image Editor";
    int w = (int)form2.numericUpDown1.Value,
      h = (int)form2.numericUpDown2.Value;
    Image im = new Bitmap(w, h);
    Graphics g = Graphics.FromImage(im);
    g.Clear(Color.White);
    g.Dispose();
    if (pictureBox1.Image != null)
      pictureBox1.Image.Dispose();
    pictureBox1.Image = im;
  }
}
private void button2_Click(object sender, EventArgs e)
{
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
    string s = openFileDialog1.FileName;
    if (pictureBox1.Image != null)
      pictureBox1.Image.Dispose();
    pictureBox1.Image = new Bitmap(s);
    Text = "Image Editor - " + s;
    saveFileDialog1.FileName = Path.ChangeExtension(s, "png");
    openFileDialog1.FileName = "";
  }
}
private void button3_Click(object sender, EventArgs e)
{
  if (saveFileDialog1.ShowDialog() == DialogResult.OK)
  {
    string s = saveFileDialog1.FileName;
    pictureBox1.Image.Save(s);
    Text = "Image Editor - " + s;
  }
}
```

Результат: в редактор можно загружать графические файлы формата BMP, JPG, PNG, GIF любого размера (кнопка **Open**). Если размер файла превосходит размер выделенной для его отображения панели `panel1`, то панель снабжается полосами прокрутки, позволяющими перемещаться к любому фрагменту изображения. При загрузке файла его полное имя отображается в заголовке окна. Загруженный файл можно сохранить под новым именем в формате PNG (кнопка **Save**). В качестве каталога для загруженных и сохраняемых файлов по умолчанию предлагается рабочий каталог приложения. Кроме того, редактор позволяет создавать новые изображения (кнопка **New**). Размер нового изображения (в пикселах) запрашивается в диалоговом окне **New Image** и может изменяться от 10 × 10 до 800 × 600. При первом отображении диалогового окна предлагаются размеры, обеспечивающие заполнение всей отображаемой части панели `panel1`; в дальнейшем в окне сохраняются размеры, указанные пользователем. Созданное изображение автоматически закрашивается белым цветом.

Недочет: при запуске программы изображение в редакторе отсутствует.

Исправление: чтобы не дублировать код, отвечающий за создание нового изображения (см. метод `button1_Click` в листинге 14.4), "делегируем" задачу создания нового изображения форме `form2`. Для этого определим обработчик события `Click` для кнопки `button1`, размещенной в форме `Form2` (листинг 14.5) и изменим метод `button1_Click` формы `Form1` (листинг 14.6; в тексте метода следует *удалить* несколько операторов, ничего не добавляя). Кроме того, в конструктор класса Form1 добавим новый оператор:

```
form2.button1_Click(this, null);
```

Листинг 14.5. Обработчик `button1.Click` (для кнопки формы `Form2`)

```
internal void button1_Click(object sender, EventArgs e)
{
  int w = (int)numericUpDown1.Value,
    h = (int)numericUpDown2.Value;
  Image im = new Bitmap(w, h);
  Graphics g = Graphics.FromImage(im);
  g.Clear(Color.White);
  g.Dispose();
  PictureBox p = Owner.Controls["panel1"].Controls["pictureBox1"]
    as PictureBox;
  if (p.Image != null)
    p.Image.Dispose();
  p.Image = im;
}
```

Листинг 14.6. Новый вариант метода `button1_Click` формы `Form1`

```
private void button1_Click(object sender, EventArgs e)
{
  form2.ActiveControl = form2.numericUpDown1;
  if (form2.ShowDialog() == DialogResult.OK)
  {
    saveFileDialog1.FileName = "";
    Text = "Image Editor";
  }
}
```

Результат: теперь при запуске программы в ней автоматически создается новое изображение, размеры которого совпадают с размерами клиентской части панели, содержащей данное изображение.

Ошибка 1: при попытке открыть файл, имеющий расширение графического файла, но содержащий данные в неверном формате (например, *пустой* файл с именем empty.bmp), возникает ошибка времени выполнения `ArgumentException` с сообщением "Parameter is not valid" (имеется в виду параметр `s` конструктора `Bitmap`). Ясно, что в подобной ситуации достаточно вывести сообщение об ошибке, однако это сообщение должно быть более информативным.

Исправление: измените метод `button2_Click` формы `Form1` (листинг 14.7).

Листинг 14.7. Новый вариант метода `button2_Click` формы `Form1`

```
private void button2_Click(object sender, EventArgs e)
{
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
    string s = openFileDialog1.FileName;
    try
    {
      Image im = new Bitmap(s);
      if (pictureBox1.Image != null)
        pictureBox1.Image.Dispose();
      pictureBox1.Image = im;
    }
    catch
    {
      MessageBox.Show("File " + s + " has a wrong format.", "Error");
      return;
    }
    Text = "Image Editor - " + s;
    saveFileDialog1.FileName = Path.ChangeExtension(s, "png");
    openFileDialog1.FileName = "";
  }
}
```

Результат: теперь при попытке загрузить файл в неверном формате выводится сообщение "File <*имя файла*> has a wrong format", причем прерывание работы программы не происходит.

Ошибка 2: при попытке сохранить загруженный файл под тем же именем (и с тем же расширением) возникает ошибка времени выполнения `ExternalException` с сообщением "A generic error occurred in GDI+". В документации .NET, посвященной методу `Save` класса `Image`, сказано, что сохранить изображение в том же файле, из которого оно было загружено, нельзя. Таким образом, выявленная ошибка связана с особенностями реализации класса `Image`.

Исправление: измените метод `button3_Click` формы `Form1` (листинг 14.8).

Листинг 14.8. Новый вариант метода `button3_Click` формы `Form1`

```
private void button3_Click(object sender, EventArgs e)
{
  string s0 = saveFileDialog1.FileName;
  if (saveFileDialog1.ShowDialog() == DialogResult.OK)
  {
    string s = saveFileDialog1.FileName;
    if (s.ToUpper() == s0.ToUpper())
    {
      s0 = Path.GetDirectoryName(s0) + "\\($$##$$).png";
      pictureBox1.Image.Save(s0);
      pictureBox1.Image.Dispose();
      File.Delete(s);
      File.Move(s0, s);
      pictureBox1.Image = new Bitmap(s);
    }
    else
      pictureBox1.Image.Save(s);
    Text = "Image Editor - " + s;
  }
}
```

Результат: теперь при попытке сохранить изображение под тем же именем (и в том же каталоге) вначале выполняется сохранение изображения во временном файле, после чего объект `Bitmap`, связанный с данным изображением, разрушается с помощью метода `Dispose`, исходный файл с изображением удаляется с диска (методом `Delete` класса `File` из пространства имен `System.IO`), а имя временного файла заменяется на имя исходного (методом `Move` класса `File`). После этого в компонент `pictureBox1` загружается новый вариант файла. Подчеркнем, что до разрушения объекта `Bitmap` связанный с ним файл удалить с диска нельзя, так как доступ к этому файлу заблокирован.

14.2. Отслеживание текущих координат изображения

Разместите в форме `Form1` метку `label1` и настройте ее свойства (листинг 14.9) и положение (рис. 14.3; ширина метки должна быть достаточной для того, чтобы отобразить трехзначные координаты вместе с комментарием **X,Y:**).

Определите обработчик события `MouseMove` для компонента `pictureBox1` (листинг 14.10).

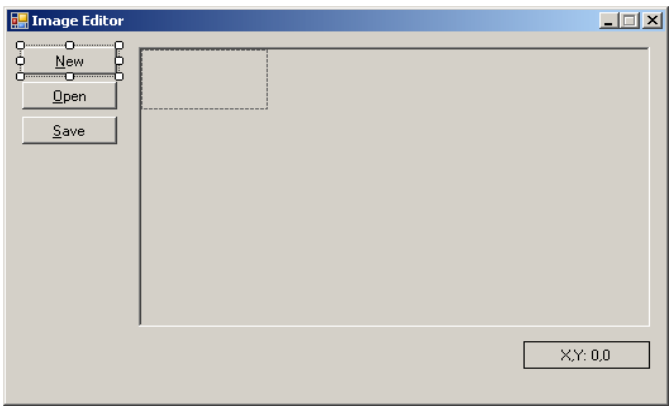

Рис. 14.3. Вид формы Form1 для проекта PNGEDIT1 на промежуточном этапе разработки

Листинг 14.9. Настройка свойств компонента `label1` формы `Form1`

```
label1: Text = X,Y: 0,0,AutoSize = False,
  BorderStyle = FixedSingle,
  TextAlign = MiddleCenter
```

Листинг 14.10. Обработчик `pictureBox1.MouseMove`

```
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
  label1.Text = string.Format("X,Y: {0},{1}", e.X, e.Y);
}
```

Результат: при перемещении мыши над изображением (то есть над компонентом `pictureBox1`) в метке `label1` отображаются координаты текущего пиксела изображения.

Примечание

Следует подчеркнуть, что выводятся именно координаты текущей точки изображения, а не позиции, отсчитываемой от левого верхнего угла компонента `panel1`. Различие между этими величинами проявляется при прокрутке изображения, имеющего размеры, которые превышают размеры панели: в подобной ситуации левый верхний угол панели будет содержать точку изображения с координатами, отличными от (0, 0).

14.3. Рисование тонким пером

Добавьте в класс `Form1` описания новых полей:

```
private Pen pen = Pens.Black;
private Point startPt;
```

Определите обработчик события `MouseDown` для компонента `pictureBox1` (листинг 14.11) и дополните метод `pictureBox1_MouseMove` (листинг 14.12).

Листинг 14.11. Обработчик `pictureBox1.MouseDown`

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

Листинг 14.12. Новый вариант метода `pictureBox1_MouseMove`

```
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
  label1.Text = string.Format("X,Y: {0},{1}", e.X, e.Y);
  if (e.Button == MouseButtons.Left)
  {
    Graphics g = Graphics.FromImage(pictureBox1.Image);
    g.DrawLine(pen, startPt, e.Location);
    g.Dispose();
    startPt = e.Location;
    pictureBox1.Invalidate();
  }
}
```

Результат: на изображении можно рисовать линии черного цвета толщиной 1 пиксел. Для этого надо переместить курсор на начальную позицию, нажать левую кнопку мыши и, не отпуская ее, нарисовать нужную линию. Рисование возможно как на созданных, так и на загруженных изображениях.

Ошибка: теперь при попытке загрузить некоторые bmp-файлы (например, файл Штукатурка.bmp, размещенный в системном каталоге Windows), возникает ошибка времени выполнения с сообщением "A Graphics object cannot be created from an image that has an indexed pixel format" ("Объект Graphics не может быть создан для изображения с индексированным форматом"). Подобная ошибка возникает для изображений, хранящихся в специальных *индексированных форматах* (в таких форматах цвета определяются не по их "абсолютному" RGB-значению, а по их индексу в специальной палитре цветов, включенной в файл), а также для изображений в монохромном формате GrayScale (использующем 256 оттенков серого цвета). Не обсуждая причины, по которым класс `Bitmap` ведет себя подобным образом, внесем в нашу программу добавление, позволяющее избежать в указанных ситуациях ошибки времени выполнения.

Исправление: в методе `button2_Click` формы `Form1` (см. листинг 14.7) вставьте после оператора

```
Image im = new Bitmap(s);
```

два следующих оператора:

```
Graphics g = Graphics.FromImage(im);
g.Dispose();
```

Результат: теперь файлы, содержащие изображения, для которых невозможно создать объект `Graphics`, нельзя загрузить в программу: при попытке их загрузки выводится такое же сообщение, как и при попытке загрузить файл в неверном формате (см. описание ошибки 1 и ее исправление в разд. 14.1).

Недочет: если при открытии или сохранении изображения выбрать файл в диалоговом окне, выполнив двойной щелчок на его имени, то после закрытия диалогового окна на изображении может появиться линия, соединяющая экранную точку, на которой был выполнен двойной щелчок, и точку, координаты которой хранятся в поле `startPt`. Этот недочет объясняется тем, что при подобном закрытии диалогового окна в компоненте `pictureBox1` может возникать событие `MouseMove`. Мы исправим этот недочет в следующей версии нашего редактора (см. разд. 15.3).

14.4. Очистка изображения

Разместите в форме `Form1` еще одну кнопку (`button4`), положите ее свойство `Text` равным **`&Clear`** (рис. 14.4) и определите обработчик события `Click` для этой кнопки (листинг 14.13).

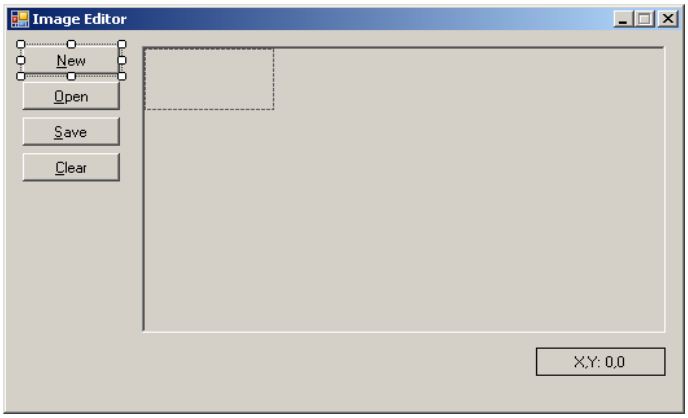

Рис. 14.4. Окончательный вид формы Form1 для проекта PNGEDIT1

Листинг 14.13. Обработчик `button4.Click`

```
private void button4_Click(object sender, EventArgs e)
{
  using (Graphics g = Graphics.FromImage(pictureBox1.Image))
    g.Clear(Color.White);
  pictureBox1.Invalidate();
}
```

Результат: при нажатии на кнопку **Clear** изображение очищается, то есть закрашивается белым цветом.

Глава 15. Цветное перо и прямые линии: проект PNGEDIT2

15.1. Рисование цветным пером

В качестве заготовки для проекта PNGEDIT2 следует использовать ранее разработанный проект PNGEDIT1 (см. гл. 14). Скопируйте проект PNGEDIT1 в новый каталог PNGEDIT2 и выполните действия, необходимые для переименования проекта (см. разд. 1.1). Разместите в форме `Form1` две новые метки (`label2` и `label3`), компонент типа `NumericUpDown` (он получит имя `numericUpDown1`) и невизуальный компонент типа `ColorDialog` (он получит имя `colorDialog1` и будет размещен под изображением формы). Настройте свойства добавленных визуальных компонентов (листинг 15.1) и расположите их в соответствии с рис. 15.1.

Измените описание поля `pen` в классе `Form1`:

```
private Pen pen = new Pen(Color.Black);
```

Определите обработчики событий `Click` и `BackColorChanged` для метки `label2`, а также обработчик события `ValueChanged` для компонента `numericUpDown1` (листинг 15.2).

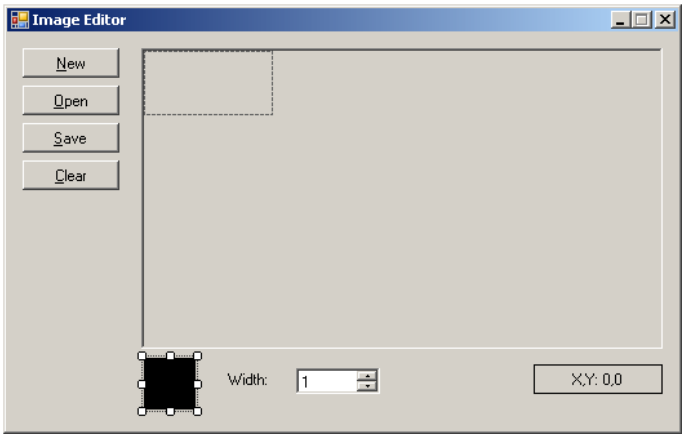

Рис. 15.1. Вид формы Form1 для проекта PNGEDIT2 на начальном этапе разработки

Листинг 15.1. Настройка свойств

```
label2: Text = пустая строка, AutoSize = False,
  Size = 40; 40, BackColor = Black,
  BorderStyle = FixedSingle
label3: Text = Width:
numericUpDown1: Maximum = 22, Minimum = 1,
  Value = 1, Increment = 3
```

Листинг 15.2. Обработчики `label2.Click`, `label2.BackColorChanged` и `numericUpDown1.ValueChanged`

```
private void label2_Click(object sender, EventArgs e)
```

```
{
  colorDialog1.Color = label2.BackColor;
  if (colorDialog1.ShowDialog() == DialogResult.OK)
    label2.BackColor = colorDialog1.Color;
}
private void label2_BackColorChanged(object sender, EventArgs e)
{
  pen.Color = label2.BackColor;
}
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
  pen.Width = (int)numericUpDown1.Value;
}
```

Результат: при рисовании можно выбирать:

- *цвет пера*, щелкая мышью на метке `label2`, фон которой соответствует текущему цвету пера; в результате на экране появляется диалоговое окно **Цвет**, позволяющее выбрать требуемый цвет (при закрытии данного окна кнопкой **OK** фон метки `label2` закрашивается выбранным цветом);
- *толщину пера*, устанавливая ее значение с помощью компонента `numericUpDown1` (допустимыми являются значения от `1` до `22`).

Установив большую толщину и белый цвет линии, можно стирать элементы изображения. При создании или загрузке нового изображения настройки пера сохраняются.

Рис. 15.2. Пример линии большой толщины до исправления недочета (слева) и после исправления (справа)

Недочет: при рисовании линий большой толщины результат оказывается неудовлетворительным (рис. 15.2, *слева*).

Исправление: в начало файла Form1.cs добавьте оператор

```
using System.Drawing.Drawing2D;
```

В конструктор класса `Form1` добавьте оператор

```
pen.StartCap = pen.EndCap = LineCap.Round;
```

Результат: теперь толстые линии рисуются надлежащим образом (рис. 15.2, *справа*).

15.2. Второй режим рисования: прямые линии

Разместите в форме компонент-контейнер типа `GroupBox` (он получит имя `groupBox1`) и присвойте его свойству `Text` значение **`Mode`**. В компоненте `groupBox1` разместите две радиокнопки (`radioButton1` и `radioButton2`) и присвойте их свойствам `Text` значения **`&Pen`** и **`&Ruler`** соответственно (рис. 15.3). Кроме того, свойство `Checked` радиокнопки `radioButton1` положите равным **`True`**.

В класс `Form1` добавьте три новых поля:

```
private int mode;
private Point movePt;
private Point nullPt = new Point(int.MaxValue, 0);
```

Определите в классе `Form1` новый вспомогательный метод `ReversibleDraw` (листинг 15.3) и обработчик события `CheckedChanged` для радиокнопки `radioButton1` (листинг 15.4), после чего свяжите созданный обработчик с событием `CheckedChanged` радиокнопки `radioButton2`.

Определите обработчик события `MouseUp` для компонента `pictureBox1` (листинг 15.5) и измените методы `pictureBox1_MouseDown` и `pictureBox1_MouseMove` (листинг 15.6).

Наконец, в начало методов `button2_Click` (листинг 14.7) и `button3_Click` (листинг 14.8) добавьте оператор

```
startPt = nullPt;
```

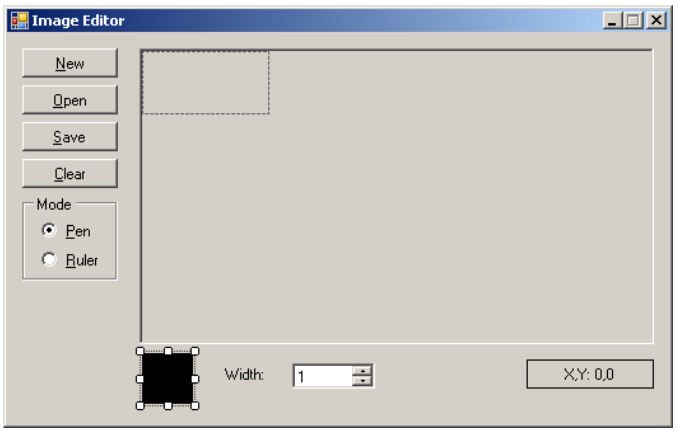

Рис. 15.3. Окончательный вид формы Form1 для проекта PNGEDIT2

Листинг 15.3. Метод `ReversibleDraw` класса `Form1`

```
private void ReversibleDraw()
{
  Point p1 = pictureBox1.PointToScreen(startPt),
    p2= pictureBox1.PointToScreen(movePt);
  ControlPaint.DrawReversibleLine(p1, p2, Color.Black);
}
```

Листинг 15.4. Обработчик `radioButton1.Click`

```
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
  RadioButton rb = sender as RadioButton;
  if (!rb.Checked)
    return;
  mode = rb.TabIndex;
}
```

Листинг 15.5. Обработчик `pictureBox1.MouseUp`

```
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
  if (startPt == nullPt)
    return;
  if (mode == 1)
  {
    Graphics g = Graphics.FromImage(pictureBox1.Image);
    g.DrawLine(pen, startPt, movePt);
    g.Dispose();
    pictureBox1.Invalidate();
  }
}
```

Листинг 15.6. Новый вариант методов `pictureBox1_MouseDown` и `pictureBox1_MouseMove`

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

```
            break;
        }
}
```

Результат: теперь рисование можно выполнять в двух режимах: в режиме **Pen** (Перо), как раньше, рисуются линии произвольной формы, а в новом режиме **Ruler** (Линейка) рисуются прямые линии. Рисование прямых линий производится следующим образом: левая кнопка мыши нажимается в начальной точке линии и затем, при нажатой кнопке, мышь перемещается к конечной точке, где кнопка мыши отпускается. При перемещении мыши на рисунке изображается текущее положение линии (для этого используется вспомогательная линия толщины 1, цвет которой является инверсным по отношению к цвету под линией). Окончательно (текущим цветом и текущей толщиной) линия рисуется в момент отпускания кнопки мыши. Для рисования ломаной линии надо после отпускания кнопки мыши сразу нажать ее еще раз, и повторять этот процесс для каждого звена ломаной. Переключение между режимами рисования осуществляется радиокнопками `radioButton1` и `radioButton2`: радиокнопка, соответствующая текущему режиму, является выбранной. Для переключения между режимами рисования можно использовать клавиши-ускорители: <Alt>+<P> (перо) и <Alt>+<R> (линейка).

Попутно исправлен недочет, выявленный в прежнем варианте программы (см. разд. 14.3). Теперь события `MouseMove` и `MouseUp`, возникающие для компонента `pictureBox1` сразу после закрытия диалогового окна `openFileDialog1` или `saveFileDialog1` (и приводящие в прежнем варианте программы к рисованию "лишней" линии), распознаются, благодаря особому значению поля `startPt`, равному `nullPt`, и в этой ситуации на изображении ничего не рисуется.

Недочет: если при рисовании линии в режиме **Ruler** вывести мышь за границы компонента `pictureBox1`, то текущая линия также будет выходить за границы данного компонента (и даже за границы окна программы), причем при отпускании кнопки мыши часть этой линии, расположенная вне компонента `pictureBox1`, останется на экране. Подобное поведение объясняется тем, что использованный метод `DrawReversibleLine` обеспечивает рисование именно на экране, не привязываясь ни к каким визуальным компонентам.

Исправление: поскольку особенности метода `DrawReversibleLine` не позволяют запретить рисование вне компонента `pictureBox1`, необходимо обеспечить, по крайней мере, удаление всех лишних линий при завершении рисования. Для этого в методе `pictureBox1_MouseUp` (см. листинг 15.5) перед оператором

```
Graphics g = Graphics.FromImage(pictureBox1.Image);
```

вставьте оператор

```
ReversibleDraw();
```

Результат: теперь в режиме **Ruler** при отпускании кнопки мыши часть линии, выходящая за границы компонента `pictureBox1`, стирается. Правда, исключением является компонент `label1` (с информацией о текущем положении курсора мыши), на котором след от линии все же может остаться. Это связано с тем, что перерисовка компонента `label1` выполняется и при рисовании линии, поэтому вызов метода `ReversibleDraw` в методе `pictureBox1_MouseUp` не сотрет, а, наоборот, нарисует линию на метке `label1`. На данный недочет можно не обращать внимания, поскольку компонент `label1` обновляется очень часто, и уже первая его перерисовка сотрет "лишнюю" линию. Можно, однако, исправить и этот недочет: достаточно после указанного выше оператора `ReversibleDraw()` добавить еще один оператор: `label1.Invalidate()`, который заставит компонент `label1` *немедленно* перерисовать себя и тем самым стереть оставшийся след от линии.

Глава 16. Прямоугольники и эллипсы, режим прозрачности: проект PNGEDIT3

16.1. Настройка фонового цвета

В качестве заготовки для проекта PNGEDIT3 следует использовать ранее разработанный проект PNGEDIT2 (см. гл. 15). Разместите в форме `Form1` новую метку `label4` и настройте ее свойства (листинг 16.1).

Используя кнопку **Send to Back**  (это вторая справа кнопка на панели **Layout** — см. разд. 9.1), переместите левую верхнюю часть метки `label4` под метку `label2`, как указано на рис. 16.1.

В класс `Form1` добавьте новое поле:

```
private SolidBrush brush = new SolidBrush(Color.White);
```

Определите обработчик события `BackColorChanged` для метки `label4` (листинг 16.2) и измените методы `button4_Click` и `label2_Click` (листинг 16.3).

Кроме того, свяжите измененный обработчик `label2_Click` с событием `Click` метки `label4`.

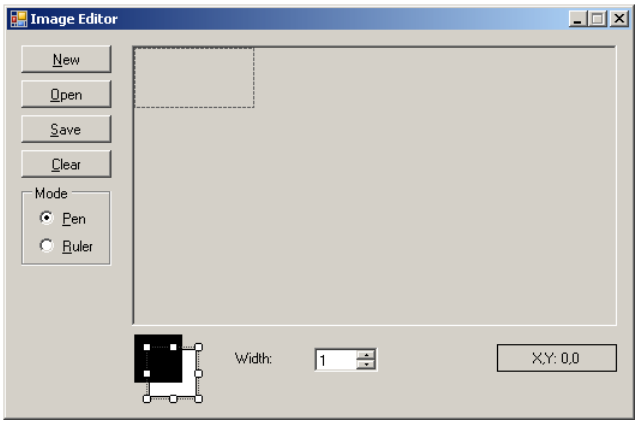

Рис. 16.1. Вид формы Form1 для проекта PNGEDIT3 на начальном этапе разработки

Листинг 16.1. Настройка свойств

```
label4: Text = пустая строка, AutoSize = False,
  Size = 40; 40, BackColor = White,
  BorderStyle = FixedSingle
```

Листинг 16.2. Обработчик label4.BackColorChanged

```
private void label4_BackColorChanged(object sender, EventArgs e)
{
  brush.Color = label4.BackColor;
}
```

Листинг 16.3. Новый вариант методов button4_Click и label2_Click

```
private void button4_Click(object sender, EventArgs e)
{
  using (Graphics g = Graphics.FromImage(pictureBox1.Image))
    g.Clear(brush.Color);
  pictureBox1.Invalidate();
}
private void label2_Click(object sender, EventArgs e)
{
  Label lb = sender as Label;
  colorDialog1.Color = lb.BackColor;
  if (colorDialog1.ShowDialog() == DialogResult.OK)
    lb.BackColor = colorDialog1.Color;
}
```

Результат: добавленная на форму метка label4 указывает текущий фоновый цвет, то есть цвет инструмента рисования *кисть* (brush), используемого для заливки внутренней области замкнутых фигур. Этот цвет можно изменить с помощью диалогового окна **Цвет**, вызвав его на экран щелчком мыши на метке label4. Новое значение фонового цвета проявляется пока только при выполнении команды **Clear**: закрашивание рисунка теперь проводится не белым, а текущим фоновым цветом. При создании или загрузке из файла нового изображения текущее значение фонового цвета сохраняется.

16.2. Третий режим рисования: прямоугольники

В компоненте groupBox1 разместите еще одну радиокнопку (radioButton3) и присвойте ее свойству Text значение **&Figure**. Кроме того, свяжите обработчик radioButton1_Click (см. листинг 15.4) с событием Click добавленной радиокнопки radioButton3.

В класс Form1 добавьте два новых метода: DrawFigure и PtToRect (листинг 16.4) и измените метод ReversibleDraw (листинг 16.5).

В методе pictureBox1_MouseMove (см. листинг 15.6) после строки

```
case 1:
```

добавьте строку

```
case 2:
```

и измените метод pictureBox1_MouseUp (листинг 16.6).

Листинг 16.4. Методы DrawFigure и PtToRect класса Form1

```
private void DrawFigure(Rectangle r, Graphics g)
{
  g.FillRectangle(brush, r);
  g.DrawRectangle(pen, r);
}
private Rectangle PtToRect(Point p1, Point p2)
{
```

```
  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);
}
```

Листинг 16.5. Новый вариант метода ReversibleDraw

```
private void ReversibleDraw()
{
  Point p1 = pictureBox1.PointToScreen(startPt),
    p2= pictureBox1.PointToScreen(movePt);
  if (mode == 1)
    ControlPaint.DrawReversibleLine(p1, p2, Color.Black);
  else
    ControlPaint.DrawReversibleFrame(PtToRect(p1, p2), Color.Black, FrameStyle.Thick);
}
```

Листинг 16.6. Новый вариант метода pictureBox1_MouseUp

```
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
  if (startPt == nullPt)
    return;
  if (mode == 1)
  {
    Graphics g = Graphics.FromImage(pictureBox1.Image);
    switch (mode)
    {
      case 1:
        g.DrawLine(pen, startPt, movePt);
        break;
      case 2:
        DrawFigure(PtToRect(startPt, movePt), g);
        break;
    }
    g.Dispose();
    pictureBox1.Invalidate();
  }
}
```

Результат: в новом режиме, который устанавливается выбором радиокнопки **Figure** (Фигура) или клавиатурной комбинацией <Alt>+<F>, можно рисовать прямоугольники. Способ рисования похож на рисование прямой линии: надо установить курсор мыши в позицию одного из углов прямоугольника, нажать левую кнопку мыши и переместить курсор в позицию противоположного по диагонали угла прямоугольника (при перемещении мыши контур прямоугольника рисуется инверсным цветом). После отпускания кнопки мыши прямоугольник окончательно рисуется текущим цветом линии, а его внутренность закрашивается текущим фоновым цветом. Ширина границы, как и толщина линии в режимах **Pen** и **Ruler**, определяется значением компонента numericUpDown1.

Недочет: если толщина пера превышает 1, то полученный прямоугольник будет иметь размеры, превышающие размеры инверсного контура, рисуемого при перемещении мыши. Это связано с тем, что по умолчанию "толстый" контур замкнутых фигур рисуется по обе стороны от их границы.

Исправление: в конструктор класса Form1 добавьте оператор

```
pen.Alignment = PenAlignment.Inset;
```

Результат: теперь рисование контура выполняется внутри границы фигуры; таким образом, инверсный контур показывает правильные размеры фигуры независимо от текущей толщины пера.

16.3. Рисование эллипсов

Разместите в форме новую метку label5, очистите ее свойство Text и положите свойство AutoSize равным **False**, а свойство TextAlign равным **MiddleCenter**. Метка label5 должна располагаться в левом нижнем углу формы (см. рис. 16.2, на котором метка label5 является текущим компонентом, то есть обрамляется белыми маркерами).

В класс Form1 добавьте новое поле:

```
private int figureMode;
```

Измените методы DrawFigure и ReversibleDraw (листинг 16.7) и определите обработчики событий Paint и MouseDown для метки label5 (листинг 16.8).

Кроме того, в методы numericUpDown1_ValueChanged, label2_BackColorChanged (см. листинг 15.2) и label4_BackColorChanged (см. листинг 16.2) добавьте оператор

```
label5.Invalidate();
```

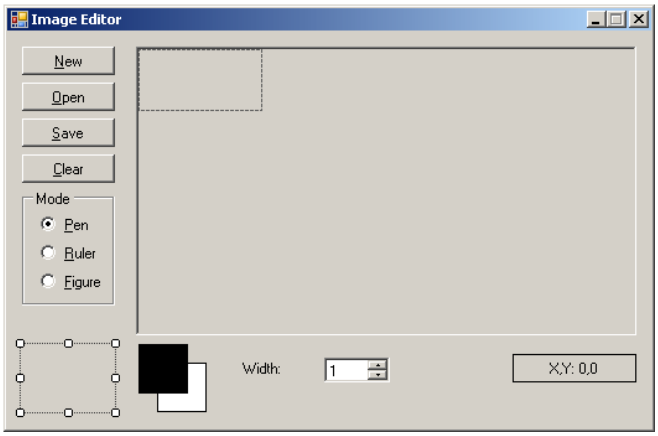

Рис. 16.2. Вид формы Form1 для проекта PNGEDIT3 на промежуточном этапе разработки

Листинг 16.7. Новый вариант методов `DrawFigure` и `ReversibleDraw`

```
private void DrawFigure(Rectangle r, Graphics g)
{
  switch (figureMode)
  {
    case 0:
      g.FillRectangle(brush, r);
      g.DrawRectangle(pen, r);
      break;
    case 1:
      g.FillEllipse(brush, r);
      g.DrawEllipse(pen, r);
      break;
  }
}
private void ReversibleDraw()
{
  Point p1 = pictureBox1.PointToScreen(startPt),
    p2= pictureBox1.PointToScreen(movePt);
  if (mode == 1)
    ControlPaint.DrawReversibleLine(p1, p2, Color.Black);
  else
    ControlPaint.DrawReversibleFrame(PtToRect(p1, p2), Color.Black, (FrameStyle)((figureMode + 1) % 2));
}
```

Листинг 16.8. Обработчики `label5.Paint` и `label5.MouseDown`

```
private void label5_Paint(object sender, PaintEventArgs e)
{
  Graphics g = e.Graphics;
  Rectangle r = label5.ClientRectangle;
  DrawFigure(r, g);
}
private void label5_MouseDown(object sender, MouseEventArgs e)
{
  radioButton3.Checked = true;
  figureMode = (figureMode + 1) % 2;
  label5.Invalidate();
}
```

Результат: при щелчке мышью на метке-образце `label5` изменяется нарисованная на ней фигура (прямоугольник переходит в эллипс, эллипс — в прямоугольник) и происходит автоматический переход в режим **Figure**. В этом режиме теперь рисуется та фигура, которая изображена на метке-образце (для рисования контура эллипса и его заливки используются методы `DrawEllipse` и `FillEllipse` класса `Graphics`). Метка `label5` также отображает текущие характеристики инструментов рисования: цвет и толщину пера и цвет кисти. При рисовании эллипсов используются тонкие штриховые инверсные контуры (в отличие от более толстых сплошных инверсных контуров, отображаемых при рисовании прямоугольников).

Недочет: если толщина пера равна 1, то в метке-образце `label5` не отображается правая и нижняя граница прямоугольника (а также правый и нижний фрагмент эллипса). Это объясняется имеющейся в GDI+ так называемой *ошибкой смещения на 1 пиксел*, которая, в частности, проявляется в том, что контуры прямоугольников и эллипсов рисуются не в прямоугольной области, указанной в качестве второго параметра методов `DrawRectangle` и `DrawEllipse`, а в области, большей указанной на 1 пиксел (по обоим измерениям).

Исправление: в методе `label5_Paint` (см. листинг 16.8) после оператора

```
Rectangle r = label5.ClientRectangle;
```

добавьте следующую строку:
```
r.Width--; r.Height--;
```

16.4. Рисование прозрачных фигур

Разместите в форме `Form1` флажок `checkBox1` и положите его свойство `Text` равным **`Transparent:`**, а свойство `RightToLeft` равным **`Yes`**. Измените свойство `Text` метки `label5` на **`Transp.`** (см. рис. 16.3).

Определите обработчик события `CheckedChanged` для флажка `checkBox1` (листинг 16.9) и измените метод `DrawFigure` (листинг 16.10).

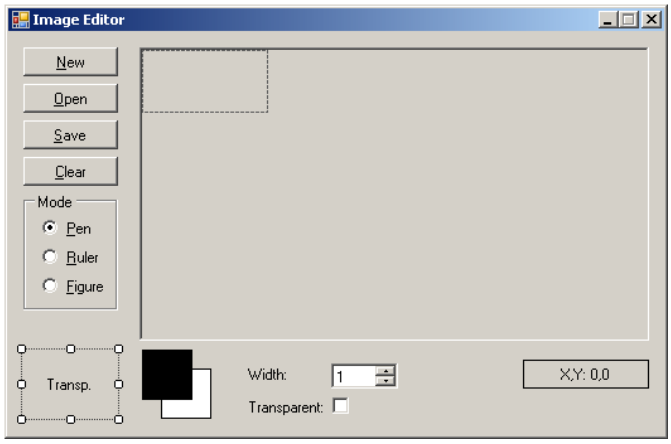

Рис. 16.3. Окончательный вид формы Form1 для проекта PNGEDIT3

Листинг 16.9. Обработчик `checkBox1.CheckedChanged`

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
  label5.Invalidate();
}
```

Листинг 16.10. Новый вариант метода `DrawFigure`

```
private void DrawFigure(Rectangle r, Graphics g)
{
  switch (figureMode)
  {
    case 0:
      if (!checkBox1.Checked)
        g.FillRectangle(brush, r);
      g.DrawRectangle(pen, r);
      break;
    case 1:
      if (!checkBox1.Checked)
        g.FillEllipse(brush, r);
      g.DrawEllipse(pen, r);
      break;
  }
}
```

Результат: установке флажка **Transparent** (Прозрачный) во включенное состояние внутренность рисуемых прямоугольников и эллипсов не закрашивается цветом фона. Если установлен прозрачный режим, то сквозь рисунок на метке-образце `label5` "просвечивает" ее заголовок **Transp.**. Следует заметить, что при установке прозрачного режима текущий цвет фона не изменяется; в частности, он по-прежнему используется при очистке изображения (т. е. при ее закрашивании текущим фоновым цветом) с помощью кнопки **Clear**.

> ***Примечание***
> Хотя в GDI+ имеется возможность устанавливать уровень прозрачности цвета (см. разд. 9.1), мы использовали более простой прием: при установленном режиме прозрачности *не вызываются* методы `FillRectangle` и `FillEllipse`, обеспечивающие заливку указанной фигуры.

Приведем изображение работающей программы (рис. 16.4).

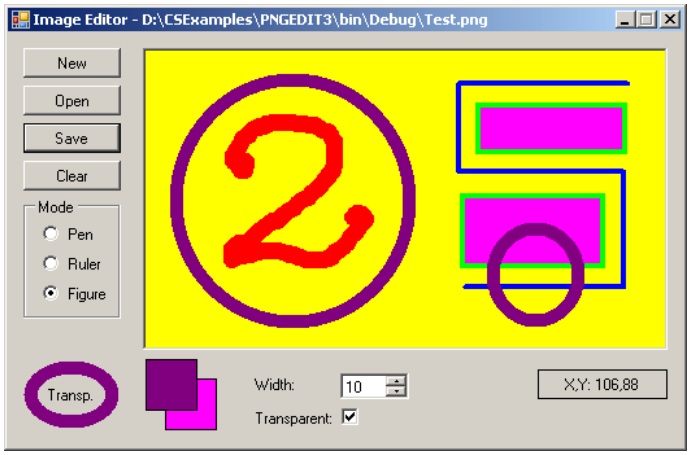

Рис. 16.4. Вид работающего приложения PNGEDIT3