Глава 18. Меню и работа с текстовыми файлами: проект TXTEDIT1

18.1. Создание меню

После создания проекта TXTEDIT1 разместите в форме `Form1` компоненты типа `MenuStrip` и `TextBox` (добавленные компоненты получат имена `menuStrip1` и `textBox1`) и настройте свойства формы и добавленных компонентов (листинг 18.1, первые две строки).

Следует отметить, что компоненты `MenuStrip` и `TextBox` надо помещать на форму в указанном порядке; в противном случае (если вначале разместить на форме `TextBox`) верхняя часть компонента `TextBox` будет скрыта под строкой меню. Впрочем, подобную ошибку легко исправить: достаточно переместить компонент `MenuStrip` на задний план кнопкой **Send to Back**  панели **Layout** (см. разд. 13.3).

Для создания меню в визуальном режиме предназначен *конструктор меню*, который активизируется при выделении компонента `menuStrip1`. В поле ввода с текстом **Type Here** следует ввести имя создаваемого пункта меню. После ввода имени и нажатия клавиши <Enter> в меню будет создан новый пункт, а рядом и под ним будут выведены новые поля ввода с текстом **Type Here**, позволяющие создать очередные пункты меню первого или второго уровня (рис. 18.1). При выделении созданного пункта меню в окне **Properties** отображаются свойства данного пункта.

Каждый пункт меню `MenuStrip` является объектом типа `ToolStripMenuItem`. Имя пункта меню, в отличие от имен других компонентов, размещаемых на форме, получается не добавлением порядкового номера к имени типа (например, `label1`), а приписыванием имени команды меню слева к имени типа, например, `fileToolStripMenuItem` (это правило действует даже для пунктов меню с русскими названиями, например, `файлToolStripMenuItem`). В результате получаются очень длинные имена, поэтому сразу после создания пункта меню желательно изменить имя пункта, указав новое значение его свойства `Name`. Заметим, что свойство `Name` располагается в окне **Properties** в *начале* списка свойств; причем подпись к нему заключается в скобки: **(Name)**.

Для того чтобы сделать более наглядной связь между пунктом меню и его именем, будем использовать в примерах стандартные английские названия пунктов меню, а в качестве их имен — эти же названия, набранные с маленькой буквы и дополненные цифрой `1` (например, для пункта **File** в качестве имени будем указывать `file1`). Цифра позволяет избежать возможных конфликтов с зарезервированными словами языка; кроме того, некоторые команды мы в дальнейшем свяжем с кнопками быстрого доступа или пунктами контекстного меню, для имен которых будет удобно использовать название той же команды, но дополненное другим номером (см. разд. 20.3 и 21.1).

Создайте в компоненте `menuStrip1` пункт меню первого уровня с текстом **`&File`** и с помощью окна **Properties** измените имя этого пункта (т. е. свойство `Name`) на **`file1`**. В выпадающем меню, связанном с пунктом **File**, создайте пункты с текстом **`&New`**, **`&Open`**, **`&Save`**, **`Save &As`**. Затем создайте пункт с текстом **`-`** (дефис); этот пункт будет преобразован в *горизонтальную разделительную линию*. Ниже разделительной линии создайте еще один пункт меню с текстом **`E&xit`**. Настройте свойства добавленных пунктов меню (листинг 18.1, строки с третьей по седьмую; заметим, что значения свойства `ShortcutKeys` проще и быстрее вводить непосредственно с клавиатуры, чем с помощью вспомогательной панели). Окончательный вид меню группы **File** приведен на рис. 18.1.

Находясь в конструкторе меню, можно также определять обработчики для событий, связанных с выделенным пунктом меню. Выделите пункт меню `exit1` и определите для него обработчик события `Click` (листинг 18.2).

Для выхода из конструктора меню достаточно выделить саму форму или какой-либо другой ее компонент.

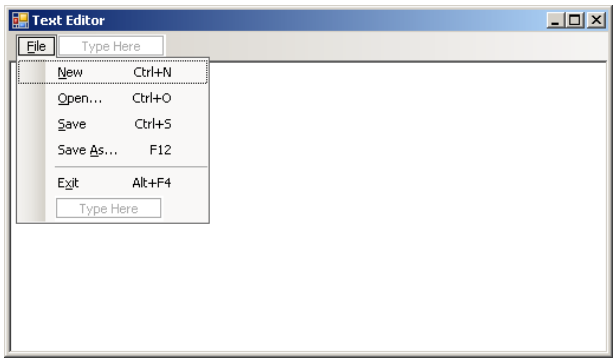

Рис. 18.1. Вид формы Form1 для проекта TXTEDIT1

Листинг 18.1. Настройка свойств

```
Form1: Text = Text Editor,
  StartPosition = CenterScreen
textBox1: Text = пустая строка, Multiline = True,
  Dock = Fill
пункт меню New (группа File): Name = new1,
  ShortcutKeys = Ctrl+N
пункт меню Open (группа File): Name = open1,
  ShortcutKeys = Ctrl+O
```

```
пункт меню Save (группа File): Name = save1,
  ShortcutKeys = Ctrl+S
пункт меню Save As (группа File): Name = saveAs1,
  ShortcutKeys = F12
пункт меню Exit (группа File): Name = exit1,
  ShortcutKeys = Alt+F4
```

Листинг 18.2. Обработчик `exit1.Click`

```
private void exit1_Click(object sender, EventArgs e)
{
  Close();
}
```

Результат: благодаря значению **Fill** свойства `Dock` компонента `textBox1`, при изменении размеров окна автоматически изменяются размеры области редактирования. В программе доступно меню, команды которого можно вызывать с помощью мыши, клавиш-*ускорителей*, называемых еще клавишами быстрого доступа, или "горячими клавишами" (<Ctrl>+<N>, <F12> и т. д.), а также с помощью клавиатурных комбинаций <Alt>+<*подчеркнутая буква в названии команды*> (если развернуто меню второго уровня, то для выбора команды достаточно нажать клавишу, соответствующую подчеркнутому символу в названии команды, не нажимая клавишу <Alt>). Команда **Exit** приводит к закрытию приложения, остальные команды меню пока не выполняют никаких действий.

18.2. Сохранение текста в файле

Добавьте в форму `Form1` невизуальный компонент типа `SaveFileDialog` (он получит имя `saveFileDialog1` и будет размещен в области невизуальных компонентов под изображением формы). Настройте свойства добавленного компонента (листинг 18.3).

В начале файла Form1.cs подключите пространство имен для файлового ввода-вывода:

```
using System.IO;
```

В конструктор класса `Form1` добавьте оператор, устанавливающий рабочий каталог программы в качестве начального каталога при сохранении файла:

```
saveFileDialog1.InitialDirectory = Directory.GetCurrentDirectory();
```

В класс `Form1` добавьте новый метод `SaveToFile` (листинг 18.4) и определите обработчики события `Click` для пунктов меню `saveAs1` и `save1` (листинг. 18.5).

Листинг 18.3. Настройка свойств

```
saveFileDialog1: DefaultExt = txt,
  Title = Сохранение файла,
  Filter = Text files|*.txt
```

Листинг 18.4. Метод `SaveToFile` класса `Form1`

```
private void SaveToFile(string path)
{
  StreamWriter sw = new StreamWriter(path, false, Encoding.Default);
  sw.WriteLine(textBox1.Text);
  sw.Close();
}
```

Листинг 18.5. Обработчики `saveAs1.Click` и `save1.Click`

```
private void saveAs1_Click(object sender, EventArgs e)
{
  if (saveFileDialog1.ShowDialog() == DialogResult.OK)
  {
    string path = saveFileDialog1.FileName;
    SaveToFile(path);
    Text = "Text Editor - " + Path.GetFileName(path);
  }
}
private void save1_Click(object sender, EventArgs e)
{
  string path = saveFileDialog1.FileName;
  if (path == "")
    saveAs1_Click(saveAs1, null);
  else
    SaveToFile(path);
}
```

Результат: при выполнении команды **Save As...** возникает диалоговое окно **Сохранение файла** (в качестве начального каталога выбирается рабочий каталог программы). Если указать в диалоговом окне имя файла и нажать кнопку **Сохранить** или клавишу <Enter>, то набранный в редакторе текст будет сохранен в этом файле, а имя файла появится в заголовке окна программы. По умолчанию имя файла снабжается расширением txt. При выполнении команды **Save** имя файла не запрашивается, за исключением случая, когда текст еще ни разу не сохранялся.

При выходе из диалогового окна **Сохранение файла** по нажатию кнопки **Отмена** или клавиши <Esc> сохранения текста в файле не происходит. При попытке сохранить текст под именем уже существующего файла выводится запрос на подтверждение данного действия. При указании несуществующего каталога выводится предупреждающее сообщение, причем диалоговое окно не закрывается, и ошибку можно немедленно исправить.

18.3. Очистка области редактирования и открытие нового файла

Добавьте в форму `Form1` невизуальный компонент типа `OpenFileDialog` (он получит имя `openFileDialog1` и будет размещен в области невизуальных компонентов под изображением формы). Настройте свойства добавленного компонента (листинг 18.6).

Измените последний оператор в конструкторе класса `Form1` следующим образом:

```
openFileDialog1.InitialDirectory = saveFileDialog1.InitialDirectory = Directory.GetCurrentDirectory();
```

Определите обработчики события `Click` для пунктов меню `new1` и `open1` (листинг. 18.7).

Листинг 18.6. Настройка свойств

```
openFileDialog1: DefaultExt = txt,
  FileName = пустая строка, Title = Открытие файла,
  Filter = Text files|*.txt
```

Листинг 18.7. Обработчики `new1.Click` и `open1.Click`

```
private void new1_Click(object sender, EventArgs e)
{
  textBox1.Clear();
  Text = "Text Editor";
  saveFileDialog1.FileName = "";
}
private void open1_Click(object sender, EventArgs e)
{
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
    string path = openFileDialog1.FileName;
    StreamReader sr = new StreamReader(path, Encoding.Default);
    textBox1.Text = sr.ReadToEnd();
    sr.Close();
    Text = "Text Editor - " + Path.GetFileName(path);
    saveFileDialog1.FileName = path;
    openFileDialog1.FileName = "";
  }
}
```

Результат: при выполнении команды **New** область редактирования очищается; при выполнении команды **Open...** появляется диалоговое окно **Открытие файла**, позволяющее выбрать файл для загрузки в редактор. В качестве начального каталога в окне **Открытие файла** устанавливается рабочий каталог приложения. При попытке открыть несуществующий файл выдается предупреждающее сообщение, однако диалоговое окно не закрывается, и ошибку можно немедленно исправить.

Заметим, что если строка из загруженного файла не умещается на экранной строке редактора, то она автоматически переносится на следующую экранную строку (разбиение строки выполняется на одном из символов пробела).

18.4. Запрос о сохранении изменений, внесенных в текст

В метод `SaveToFile` класса `Form1` добавьте оператор

```
textBox1.Modified = false;
```

В класс `Form1` добавьте новый метод `TextSaved` (листинг 18.8). Измените метод `new1_Click` (листинг 18.9) и добавьте в начало метода `open1_Click` новую строку:

```
if (TextSaved())
```

Кроме того, определите обработчик события `FormClosing` для формы `Form1` (листинг 18.10).

Листинг 18.8. Метод `TextSaved` класса `Form1`

```
private bool TextSaved()
{
  if (textBox1.Modified)
    switch (MessageBox.Show("Сохранить изменения в документе?", "Подтверждение",
            MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question))
    {
      case DialogResult.Yes:
        save1_Click(this, null);
        return !textBox1.Modified;
      case DialogResult.Cancel:
        return false;
    }
```

```
  return true;
}
```

Листинг 18.9. Новый вариант метода new1_Click

```
private void new1_Click(object sender, EventArgs e)
{
  if (TextSaved())
  {
    textBox1.Clear();
    Text = "Text Editor";
    saveFileDialog1.FileName = "";
  }
}
```

Листинг 18.10. Обработчик Form1.FormClosing

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
  e.Cancel = !TextSaved();
}
```

Результат: если в текущий текст были внесены изменения, то попытка очистить окно редактора командой **New**, открыть новый файл командой **Open** или выйти из программы приводит к появлению запроса о том, следует ли сохранять на диске внесенные изменения. При нажатии кнопки **Да** текущий текст сохраняется под прежним именем, а если он ни разу не был сохранен, то его имя запрашивается в диалоге **Сохранение файла**. При нажатии кнопки **Нет** изменения не сохраняются. При нажатии кнопки **Отмена** выбранное действие (команды **New**, **Open** или выход из программы) отменяется, и пользователь может продолжать редактирование текущего текста.

Глава 9. Цвета: проект COLORS

9.1. Определение цвета как комбинации четырех цветовых составляющих. Ползунки

После создания проекта COLORS разместите в форме Form1 компонент TrackBar (он получит имя trackBar1) и настройте свойства формы и компонента trackBar1 (листинг 9.1; указанный в листинге компонент label6 будет добавлен к проекту позднее).

Не снимая выделения с компонента trackBar1, нажмите комбинацию <Ctrl>+<C>, а затем четыре раза — комбинацию <Ctrl>+<V>. В результате в форму будут добавлены еще четыре компонента TrackBar с именами trackBar2–trackBar5, причем для всех этих компонентов значения свойств будут совпадать со значениями соответствующих свойств компонента trackBar1 (заметим, что подобное копирование компонентов можно выполнять и с помощью контекстного меню формы). Расположите все компоненты сверху вниз (рис. 9.1).

Для компонента trackBar1 дополнительно установите свойство Value равным **255** (у остальных компонентов TrackBar это свойство должно остаться равным **0**).

После этого разместите в форме пять меток (label1, ..., label5) и установите их свойства Text равными соответственно **Alpha**, **Red**, **Green**, **Blue**, **Gray**.

Метки надо выровнять по середине соответствующих компонентов TrackBar (см. рис. 9.1). Для этого удобно использовать дополнительное средство: *панель выравнивания* **Layout**, которую можно отобразить на экране с помощью команды меню **View** | **Toolbars** | **Layout**. Для выполнения выравнивания следует вначале выделить один из компонентов TrackBar, а *затем* — соответствующую ему метку, после чего нажать на кнопку  (**Align Middles**) на панели выравнивания. Порядок выделения компонентов важен, поскольку выравнивание производится по активному компоненту (его маркеры имеют белый цвет).

Наконец, разместите в нижней части формы компонент-контейнер panel1 и в этот компонент поместите еще одну метку (label6). Настройте свойства метки label6 (листинг 9.1; значения свойств ForeColor и BackColor проще всего набрать на клавиатуре, не используя вспомогательные окна выбора цвета).

Определите обработчик события Scroll для компонента trackBar1 (листинг 9.2), после чего свяжите созданный обработчик с событиями Scroll компонентов trackBar2–trackBar4 (обработчик события Scroll для компонента trackBar5 будет добавлен позднее, в разд. 9.3).

Для компонента panel1 задайте фоновый рисунок, взяв его из набора фоновых рисунков Windows. С этой целью вначале выделите компонент panel1 в форме (поскольку компонент panel1 находится под меткой label6, проще всего выделить эту метку, после чего нажать клавишу <Esc>). Затем выделите в окне **Properties** свойство BackgroundImage компонента panel1 и нажмите на кнопку с многоточием  . В появившемся окне **Select Resource** надо нажать кнопку **Import** (если кнопка недоступна, то предварительно надо выбрать переключатель **Project resource file**) и в новом окне **Open** выбрать файл с требуемым рисунком, после чего нажать клавишу <Enter> или кнопку **Открыть**. Будем считать, что выбран файл Штукатурка.bmp из каталога Windows. После выполнения описанных действий имя выбранного файла появится в списке ресурсов и будет выделено; кроме того, в правой части окна **Select Resource** появится изображение, взятое из этого файла. Осталось закрыть окно **Select Resource**, нажав кнопку **OK**. В результате свойство BackgroundImage панели panel1 получит значение **COLORS.Properties.Resources.Штукатурка**. Кроме того, в окне проекта **Solution Explorer** появится но-

вый элемент: файл Штукатурка.bmp, размещенный в разделе **Resources**. Следует заметить, что рисунок панели будет заслонен меткой `label6`, поэтому увидеть его мы пока не сможем.

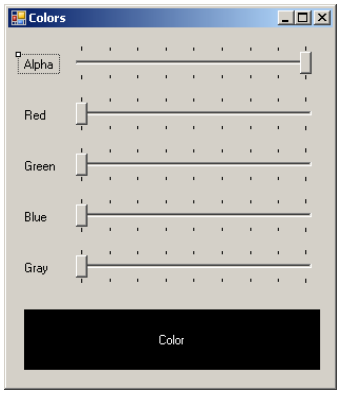

Рис. 9.1. Вид формы Form1 для проекта COLORS

Листинг 9.1. Настройка свойств

```
Form1: Text = Colors, StartPosition = CenterScreen
trackBar1: LargeChange = 32, Maximum = 255,
  TickFrequency = 32, TickStyle = Both
label6: Text = Color, AutoSize = False,
  Dock = Fill, ForeColor = White,
  BackColor = Black
```

Листинг 9.2. Обработчик `trackBar1.Scroll`

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
  label6.BackColor = Color.FromArgb(trackBar1.Value, trackBar2.Value, trackBar3.Value, trackBar4.Value);
}
```

Результат: цвет фона метки `label6` определяется как комбинация четырех цветовых составляющих, а именно, прозрачности (Alpha) и интенсивности трех базовых цветов: красного (Red), зеленого (Green) и синего (Blue). Каждая цветовая составляющая может меняться в пределах от 0 до 255; значение 255 для составляющей Alpha соответствует полной непрозрачности. В нашей программе значения цветовых составляющих задаются положением соответствующего компонента `TrackBar` (компонент `trackBar5` пока не используется). Поскольку панель с меткой `label6` содержит фоновый рисунок, этот рисунок "просвечивает" сквозь фон метки при уровне прозрачности, отличном от 255.

9.2. Инвертирование цветов и вывод цветовых констант

Добавьте новые операторы в метод `trackBar1_Scroll` (листинг 9.3).

Листинг 9.3. Новый вариант метода `trackBar1_Scroll`

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
  label6.BackColor = Color.FromArgb(trackBar1.Value, trackBar2.Value,
    trackBar3.Value, trackBar4.Value);
  Color c = label6.BackColor;
  label6.ForeColor = Color.FromArgb(0xFF ^ c.R, 0xFF ^ c.G, 0xFF ^ c.B);
  label6.Text = c.Name.ToUpper();
}
```

Результат: числовое значение текущего цвета в формате ARGB (Alpha–Red–Green–Blue) отображается на панели в виде шестнадцатеричного числа. При этом для каждой цветовой составляющей отводится по два знака, а буквы A–F (соответствующие шестнадцатеричным цифрам от 10 до 15) изображаются в верхнем регистре. Например, значение цвета Maroon (непрозрачный темно-красный цвет интенсивности 128) имеет вид **FF800000**. Цвет текста — непрозрачный и инверсный по отношению к цвету фона панели.

Недочет 1: при запуске программы метка `label6` содержит текст **Color**, а не числовое значение непрозрачного черного цвета.

Исправление: в конструктор класса `Form1` добавьте оператор:

```
trackBar1_Scroll(null, null);
```

Результат: теперь метод `trackBar1_Scroll` вызывается в момент создания формы, что обеспечивает правильную настройку внешнего вида метки `label6` (при вызове метода оба его параметра можно положить равными `null`, поскольку в этом методе ни один из параметров не используется).

Недочет 2: если прозрачность имеет значение, меньшее `16`, то в метке `label6` отображается менее 8 цифр (в частности, при полностью прозрачном черном цвете метка будет содержать единственную цифру **0**). Подобный способ представления цвета является не вполне естественным; логичнее *всегда* отображать шестнадцатеричное число с 8 знаками (по два знака на каждую цветовую составляющую).

Исправление: в методе `trackBar1_Scroll` замените последний оператор на следующий:

```
label6.Text = c.ToArgb().ToString("X8");
```

Результат: теперь число `c.ToArgb()` выводится в шестнадцатеричном формате `X` с заглавными латинскими буквами и обязательными 8 знаками (мы использовали форматирующий вариант метода `ToString` для типа `int`).

9.3. Отображение оттенков серого цвета

Определите обработчик события `Scroll` для компонента `trackBar5` (листинг 9.4).

Листинг 9.4. Обработчик `trackBar5.Scroll`

```
private void trackBar5_Scroll(object sender, EventArgs e)
{
  trackBar2.Value = trackBar3.Value = trackBar4.Value = trackBar5.Value;
}
```

Результат: перемещение ползунка `trackBar5` обеспечивает синхронное изменение всех трех базовых цветов, давая в итоге различные оттенки серого цвета (значение прозрачности при этом не изменяется).

Ошибка: несмотря на то, что при перемещении ползунка `trackBar5` положение ползунков `trackBar2`—`trackBar4` синхронно изменяется, это изменение не влияет на цвет метки `label6`. Отмеченная ошибка объясняется тем обстоятельством, что при *программном* изменении свойства `Value` событие `Scroll` не возникает.

Исправление: добавьте в метод `trackBar5_Scroll` оператор:

```
trackBar1_Scroll(null, null);
```

9.4. Вывод цветовых имен

Измените метод `trackBar1_Scroll` (листинг 9.5).

Листинг 9.5. Новый вариант метода `trackBar1_Scroll`

```
private void trackBar1_Scroll(object sender, EventArgs e)
{
  label6.BackColor = Color.FromArgb(trackBar1.Value, trackBar2.Value,
    trackBar3.Value, trackBar4.Value);
  Color c = label6.BackColor;
  label6.ForeColor = Color.FromArgb(0xFF ^ c.R, 0xFF ^ c.G, 0xFF ^ c.B);
  string s = c.ToArgb().ToString("X8");
  switch (c.A)
  {
    case 0:
      s += " Transparent";
      break;
    case 255:
      string
        s0 = ColorTranslator.FromWin32(ColorTranslator.ToWin32(c)).Name;
      if (s0.Substring(0, 2) != "ff")
        s += " " + s0;
      break;
  }
  label6.Text = s;
}
```

Результат: в том случае, когда с текущим цветом связано определенное имя (например, **Black** или **Maroon**), на панели отображается не только числовое значение текущего цвета в шестнадцатеричном формате, но и его имя. Если прозрачность имеет значение, равное 0, то рядом с числовым значением цвета выводится текст **Transparent**.

9.5. Связывание компонентов с подписями к ним

Измените свойства `Text` для меток `label1`–`label5`, добавив в них символ `&`: **`&Alpha`**, **`&Red`**, **`&Green`**, **`&Blue`**, **`Gra&y`** (для метки `Gray` символ `&` выделяет последнюю букву, поскольку все предыдущие буквы уже использованы в качестве выделенных символов других меток).

Используя команду меню **View | Tab Order**, установите значения свойства `TabIndex` для компонентов, размещенных в форме: `label1` — **0**, `trackBar1` — **1**, `label2` — **2**, `trackBar2` — **3**, `label3` — **4**, `trackBar3` — **5**, `label4` — **6**, `trackBar4` — **7**, `label5` — **8**, `trackBar5` — **9**.

Результат: переключение между ползунками теперь можно осуществлять с помощью <Alt>-комбинаций символов, подчеркнутых в подписях к ползункам (<Alt>+<A> для ползунка, определяющего прозрачность, <Alt>+<R> для ползунка, определяющего интенсивность красного цвета, и т. д.). Отметим, что <Alt>-комбинации выполняют требуемое действие только при установке английской раскладки клавиатуры.

9.6. Привязка компонентов

Измените свойство `Anchor` для компонентов `trackBar1`–`trackBar5`, положив его равным **`Top, Left, Right`** (при установке свойства `Anchor` в окне **Properties** отображается специальная панель; для настройки в этой панели требуемого вари-

анта следует выделить линию, ведущую к *правой* границе панели, поскольку линии, ведущие к верхней и левой границе уже выделены по умолчанию).

Аналогичными действиями установите свойство `Anchor` компонента `panel1` равным **`Top, Bottom, Left, Right`** (в данном случае в панели настройки свойства `Anchor` надо выделить линии, ведущие к нижней и правой границам, оставив также выделенными линии, ведущие к верхней и левой границам). Подчеркнем, что необходимо настроить свойство `Anchor` *панели* `panel1`, а не содержащейся на ней метки `label6`.

Результат: если при выполнении программы изменить размер формы `Form1`, то размер компонентов будет скорректирован в соответствии с новым размером формы (ползунки `trackBar1`–`trackBar5` изменяют ширину, а панель `panel1` с меткой `label6` — ширину и высоту).

Недочет: при уменьшении размеров формы может возникнуть ситуация, когда цветовая метка перестанет быть видна, а ползунки станут слишком узкими.

Исправление: добавьте в конструктор класса `Form1` оператор

```
MinimumSize = Size;
```

Результат: Теперь размеры формы можно только увеличить (по сравнению с исходными), поскольку минимально допустимые размеры формы, определяемые свойством `MinimumSize` совпадают с исходными размерами формы, хранящимися в свойстве `Size`.

Глава 24. Флажки и группы флажков: проект CHKBOXES

24.1. Установка флажков и проверка их состояния

После создания проекта CHKBOXES разместите в форме `Form1` три флажка (`checkBox1`–`checkBox3`), три метки (`label1`–`label3`) и кнопку `button1`. Настройте свойства формы и добавленных компонентов (листинг 24.1) и расположите компоненты в соответствии с рис. 24.1. Поскольку метки `label1`-`label3` имеют одинаковое значение свойства `Text`, удобно настроить это свойство сразу для всех меток, предварительно выделив их.

Добавьте к проекту новую форму (она получит имя `Form2`) и разместите в ней три компонента — *списка флажков* типа `CheckedListBox` (они получат имена `checkedListBox1`–`checkedListBox3`) и кнопку `button1`. Настройте свойства формы `Form2` и ее компонентов (листинг 24.2) и расположите компоненты в соответствии с рис. 24.2. При определении свойства `Items` компонентов `CheckedListBox` используется специальное диалоговое окно; в нашем случае в это окно надо ввести указанные числа (5 чисел для `checkedListBox1`, 4 числа для `checkedListBox2`, 6 чисел для `checkedListBox3`), набирая каждое число на отдельной строке.

В описание класса `Form1` добавьте новое поле

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

В конструктор класса `Form1` добавьте оператор

```
AddOwnedForm(form2);
```

Определите обработчик события `Click` для кнопки `button1` формы `Form1` (листинг 24.3), а также обработчик события `FormClosed` для формы `Form2` (листинг 24.4).

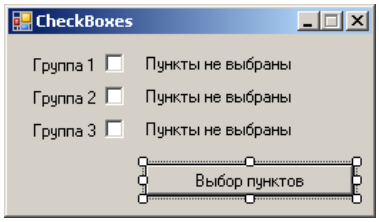

Рис. 24.1. Вид формы Form1 для проекта CHKBOXES

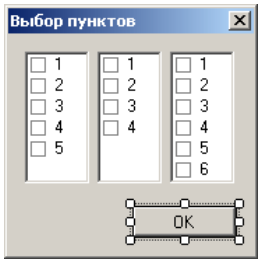

Рис. 24.2. Вид формы Form2 для проекта CHKBOXES

Листинг 24.1. Настройка свойств формы `Form1` и ее компонентов

```
Form1: Text = CheckBoxes, MaximizeBox = False,
  FormBorderStyle = FixedSingle,
  AcceptButton = button1
checkBox1: Text = Группа 1, RightToLeft = Yes
```

```
checkBox2: Text = Группа 2, RightToLeft = Yes
checkBox3: Text = Группа 3, RightToLeft = Yes
label1-label3: Text = Пункты не выбраны
button1: Text = Выбор пунктов
```

Листинг 24.2. Настройка свойств формы Form2 и ее компонентов

```
Form2: Text = Выбор пунктов, MaximizeBox = False,
  MinimizeBox = False,
  FormBorderStyle = FixedDialog,
  StartPosition = CenterScreen,
  ShowInTaskbar = False, AcceptButton = button1
checkedListBox1: Items = 1 2 3 4 5,
  CheckOnClick = true
checkedListBox2: Items = 1 2 3 4,
  CheckOnClick = true
checkedListBox3: Items = 1 2 3 4 5 6,
  CheckOnClick = true
button1: Text = OK, DialogResult = OK
```

Листинг 24.3. Обработчик button1.Click для формы Form1

```
private void button1_Click(object sender, EventArgs e)
{
  form2.ShowDialog();
}
```

Листинг 24.4. Обработчик Form2.FormClosed

```
private void Form2_FormClosed(object sender, FormClosedEventArgs e)
{
  for (int i = 1; i <= 3; i++)
  // обработка каждой группы флажков
  {
    string s = "";
    CheckedListBox clb =
      Controls["checkedListBox" + i] as CheckedListBox;
    int k = clb.CheckedIndices.Count;
    if (k == 0)
      s = "Пункты не выбраны";
    else
      if (k == clb.Items.Count)
        s = "Выбраны все пункты";
      else
      {
        foreach (int j in clb.CheckedIndices)
          s = s + " " + clb.Items[j].ToString();
        s = "Выбрано:" + s;
      }
    // вывод информации о флажках
    // в соответствующей метке формы Form1
    Owner.Controls["label" + i].Text = s;
  }
}
```

Результат: если вызвать диалоговое окно **Выбор пунктов**, установить в нем какие-либо флажки и закрыть его, то на главной форме **CheckBoxes** отобразится информация о выбранных пунктах в каждой группе. Флажки на главной форме пока не используются.

24.2. Глобальная установка флажков

Определите обработчик события CheckStateChanged для флажка checkBox1 (листинг 24.5), после чего свяжите созданный обработчик с событием CheckStateChanged флажков checkBox2 и checkBox3.

Кроме того, измените метод button1_Click формы Form1 (листинг 24.6).

Листинг 24.5. Обработчик checkBox1.CheckStateChanged

```
private void checkBox1_CheckStateChanged(object sender, EventArgs e)
{
  CheckBox cb = sender as CheckBox;
  string s = "label" + cb.Name[cb.Name.Length - 1];
  Controls[s].Text = cb.Checked ? "Выбраны все пункты" : "Пункты не выбраны";
}
```

Листинг 24.6. Новый вариант метода button1_Click формы Form1

```
private void button1_Click(object sender, EventArgs e)
{
  for (int i = 1; i <= 3; i++)
```

```
  {
    CheckedListBox clb =
      form2.Controls["checkedListBox" + i] as CheckedListBox;
    bool b = (Controls["checkBox" + i] as CheckBox).Checked;
    for (int j = 0; j < clb.Items.Count; j++)
      clb.SetItemChecked(j, b);
  }
  form2.ShowDialog();
}
```

Результат: установка флажка на главной форме обеспечивает выбор всех пунктов соответствующей группы, а его снятие обеспечивает отмену всех выбранных пунктов. При открытии диалогового окна **Выбор пунктов** его списки флажков корректируются.

Недочет: явно сделанные установки в диалоговом окне не влияют на вид флажков главной формы. Заметим, что если в диалоговом окне выбрана часть пунктов, то флажок соответствующей группы на главной форме принято изображать затененным (то есть находящимся в состоянии `CheckState.Indeterminate`). Необходимые исправления будут сделаны в следующем разделе.

24.3. Использование флажков, принимающих три состояния

Измените метод `Form2_FormClosed` формы `Form2` (листинг 24.7).

Листинг 24.7. Новый вариант метода `Form2_FormClosed` формы `Form2`

```
private void Form2_FormClosed(object sender, FormClosedEventArgs e)
{
  for (int i = 1; i <= 3; i++)
  {
    CheckedListBox clb =
      Controls["checkedListBox" + i] as CheckedListBox;
    CheckBox cb = Owner.Controls["checkBox" + i] as CheckBox;
    int k = clb.CheckedIndices.Count;
    if (k == 0)
      cb.Checked = false;
    else
      if (k == clb.Items.Count)
        cb.CheckState = CheckState.Checked;
      else
      {
        string s = "";
        foreach (int j in clb.CheckedIndices)
          s = s + " " + clb.Items[j].ToString();
        s = "Выбрано:" + s;
        Owner.Controls["label" + i].Text = s;
        cb.CheckState = CheckState.Indeterminate;
      }
  }
}
```

Результат: в случае выбора части флажков в одном из списков окна **Выбор пунктов** соответствующий флажок в окне **CheckBoxes** становится затененным. Заметим, что пользователь не может явным образом устанавливать флажки окна **CheckBoxes** в затененное состояние, поскольку для компонентов `checkBox1`–`checkBox3` свойство `ThreeState` равно **`False`** (значение по умолчанию).

> ***Примечание***
>
> Обратите внимание на то, что в новом варианте обработчика `Form2_FormClosed` текст метки изменяется только при выборе *части пунктов*. Если выбраны все пункты или не выбрано ни одного, то изменение текста метки происходит в обработчике `checkBox1_CheckStateChanged` (см. листинг 24.5), который автоматически вызывается при любом изменении состояния флажка.

Ошибка 1: рядом с затененным флажком *всегда* отображается текст **Выбраны все пункты**. Эта ошибка связана с тем, что при установке свойства `CheckState` равным `Indeterminate` вызывается обработчик события `CheckStateChanged`, причем в этом случае свойство `Checked` имеет значение `true`.

Исправление: откорректируйте метод `checkBox1_CheckStateChanged` формы `Form1` (листинг 24.8).

Листинг 24.8. Новый вариант метода `checkBox1_CheckStateChanged` формы `Form1`

```
private void checkBox1_CheckStateChanged(object sender, EventArgs e)
{
  CheckBox cb = sender as CheckBox;
  if (cb.CheckState == CheckState.Indeterminate)
    return;
  string s = "label" + cb.Name[cb.Name.Length - 1];
  Controls[s].Text = cb.Checked ? "Выбраны все пункты" :
    "Пункты не выбраны";
}
```

Ошибка 2: при повторном вызове окна **Выбор пунктов** затененные флажки, подобно установленным, приводят к установке всех флажков в соответствующем списке (эта ошибка, как и ошибка 1, связана с тем, что свойство `Checked` имеет значение `true` как для установленных, так и для затененных флажков).

Исправление: откорректируйте метод `button1_Click` формы `Form1` (листинг 24.9).

Листинг 24.9. Новый вариант метода `button1_Click` формы `Form1`

```
private void button1_Click(object sender, EventArgs e)
{
  for (int i = 1; i <= 3; i++)
  {
    CheckedListBox clb =
      form2.Controls["checkedListBox" + i] as CheckedListBox;
    CheckState cs = (Controls["checkBox" + i] as CheckBox).CheckState;
    if (cs == CheckState.Indeterminate)
      continue;
    for (int j = 0; j < clb.Items.Count; j++)
      clb.SetItemCheckState(j, cs);
  }
  form2.ShowDialog();
}
```

Результат: теперь в случае затененных флажков связанные с ними списки флажков при открытии окна **Выбор пунктов** не изменяются. Для этого в новом варианте метода `button1_Click` используется оператор `continue`, который обеспечивает немедленное завершение текущей итерации цикла.