Глава 25. Выпадающие и обычные списки: проект LISTBOX1

25.1. Создание и использование выпадающих списков

После создания проекта LISTBOX1 разместите в форме `Form1` выпадающий список `comboBox1` и панель `panel1` (рис. 25.1). Настройте свойства формы и добавленных компонентов (листинг 25.1). Отметим, что, положив свойство `DropDownStyle` компонента `comboBox1` равным **`DropDownList`**, мы тем самым отключили возможность ввода новых значений для этого компонента (разрешено выбирать только значения из имеющегося списка).

Дополните конструктор класса `Form1` (листинг 25.2) и определите обработчик события `SelectedIndexChanged` для компонента `comboBox1` (листинг 25.3).

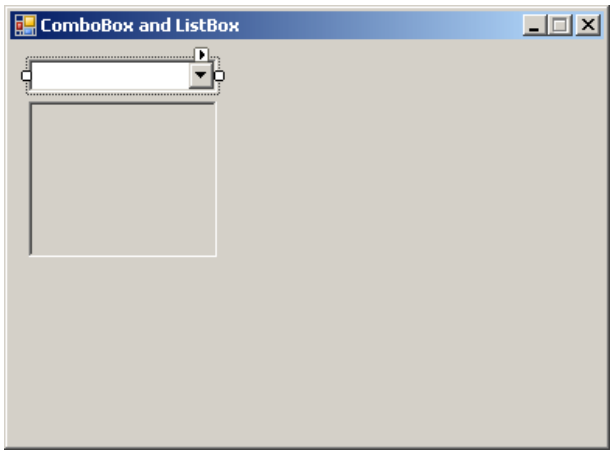

Рис. 25.1. Вид формы Form1 для проекта LISTBOX1 на начальном этапе разработки

Листинг 25.1. Настройка свойств

```
Form1: Text = ComboBox and ListBox,
  MaximizeBox = False,
  FormBorderStyle = FixedSingle,
  StartPosition = CenterScreen
comboBox1: Text = пустая строка,
  DropDownStyle = DropDownList
panel1: BorderStyle = Fixed3D
```

Листинг 25.2. Новый вариант конструктора формы `Form1`

```
public Form1()
{
  InitializeComponent();
  string[] s = Enum.GetNames(typeof(KnownColor));
  int n1 = Array.IndexOf(s, "AliceBlue"),
    cnt = Array.IndexOf(s, "YellowGreen") - n1 + 1;
  string[] s0 = new string[cnt];
  Array.Copy(s,n1,s0,0,cnt);
  comboBox1.Items.AddRange(s0);
  comboBox1.SelectedIndex = 0;
}
```

Листинг 25.3. Обработчик `comboBox1.SelectedIndexChanged`

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  panel1.BackColor = Color.FromName(comboBox1.Items[comboBox1.SelectedIndex].ToString());
}
```

Результат: при запуске программы в выпадающем списке содержатся значения идентификаторов для всех именованных цветов. Идентификаторы указаны в алфавитном порядке. Выбор идентификатора из списка приводит к соответствующему изменению цвета панели `panel1`.

25.2. Список: добавление и удаление элементов

Разместите в форме `Form1` список `listBox1`, панель `panel2`, две метки `label1` и `label2` (компонент `panel2` следует разместить под компонентом `panel1`, как показано на рис. 25.2). Настройте свойства добавленных компонентов (листинг 25.4).

Определите обработчики события `SelectedIndexChanged` для компонента `listBox1` и события `Click` для компонентов `label1` и `label2` (листинг 25.5).

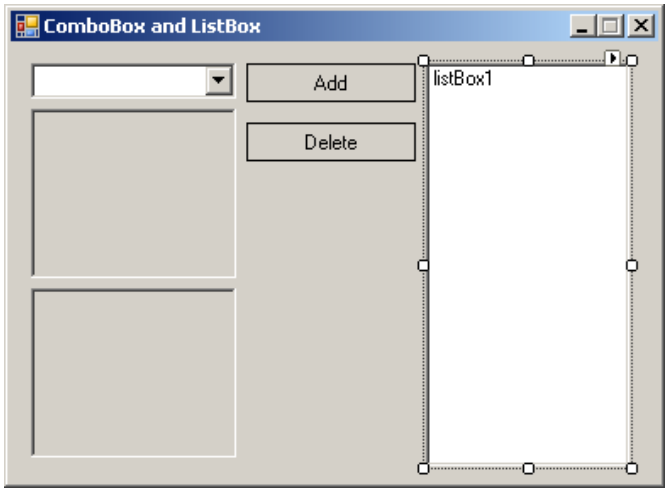

Рис. 25.2. Вид формы Form1 для проекта LISTBOX1 на промежуточном этапе разработки

Листинг 25.4. Настройка свойств

```
panel2: BorderStyle = Fixed3D
label1: Text = Add, AutoSize = False,
  TextAlign = MiddleCenter,
  BorderStyle = FixedSingle
label2: Text = Delete, AutoSize = False,
  TextAlign = MiddleCenter,
  BorderStyle = FixedSingle
```

Листинг 25.5. Обработчики `listBox1.SelectedIndexChanged`, `label1.Click` и `label2.Click`

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  panel2.Visible = listBox1.Items.Count > 0;
  if (listBox1.SelectedIndex == -1)
    return;
  panel2.BackColor = Color.FromName(listBox1.Items[listBox1.SelectedIndex].ToString());
}
private void label1_Click(object sender, EventArgs e)
{
  listBox1.SelectedIndex = listBox1.Items.Add(comboBox1.Text);
}
private void label2_Click(object sender, EventArgs e)
{
  listBox1.Items.RemoveAt(listBox1.SelectedIndex);
}
```

Результат: при щелчке на метке **Add** в список `listBox1` добавляется строка из выпадающего списка `comboBox1`. Цвет панели `panel2` соответствует выделенному элементу списка. Если список является пустым, то панель `panel2` на форме не отображается. При щелчке на метке **Delete** из списка удаляется выделенный элемент.

Недочет: после выполнения операции удаления в списке исчезает выделенный элемент (хотя элемент, обведенный рамкой, остается). В этой ситуации свойство `SelectedIndex` компонента `listBox1` равно `-1`, поэтому повторный щелчок на метке **Delete** приводит к ошибке времени выполнения (так как значение `-1` не является допустимым значением для метода `RemoveAt`). По этой же причине к ошибке приводит щелчок на метке **Delete** в случае пустого списка.

Исправление: измените метод `label2_Click` (листинг 25.6).

Листинг 25.6. Новый вариант метода `label2_Click`

```
private void label2_Click(object sender, EventArgs e)
{
  int i = listBox1.SelectedIndex;
  if (i == -1)
  {
    Console.Beep();
    return;
  }
  listBox1.Items.RemoveAt(i);
  if (i == listBox1.Items.Count)
    i--;
  listBox1.SelectedIndex = i;
}
```

Результат: При удалении элемента в середине списка выделение сохраняется на текущей позиции (которую теперь занимает следующий элемент). При удалении элемента в конце списка выделяется предыдущий элемент. Таким образом, если

список не пуст, он всегда имеет выделенный элемент. При щелчке на метке **Delete** в случае пустого списка выдается звуковой сигнал.

Примечание

Обратите внимание на то, что в методе `listBox1_SelectedIndexChanged` мы предусмотрели особую обработку ситуации `SelectedIndex = -1` (см. листинг 25.6), поскольку такая ситуация возникает в ходе удаления элемента списка.

25.3. Дополнительные операции над списком

Разместите в форме `Form1` пять новых меток (`label3-label7`) и настройте их свойства (листинг 25.7; поскольку свойства `AutoSize`, `TextAlign` и `BorderStyle` для всех меток имеют одинаковые значения, удобно установить их одновременно, предварительно выделив все метки на форме). Расположите новые метки на форме в соответствии с рис. 25.3.

В начало файла Form1.cs добавьте строку

```
using System.IO;
```

Определите обработчики события `Click` для добавленных меток (листинг 25.8).

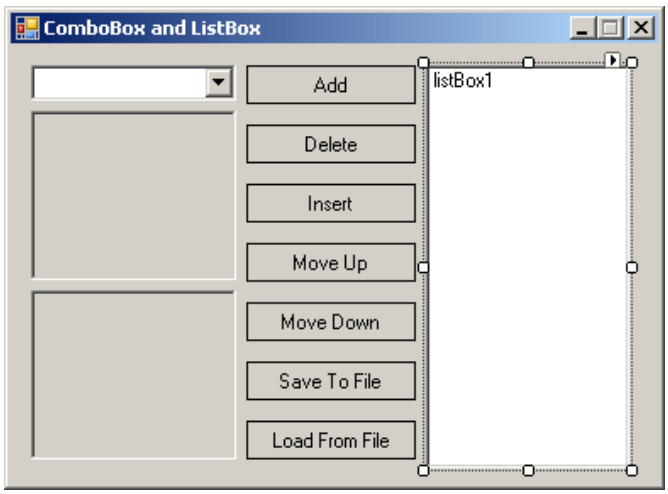

Рис. 25.3. Окончательный вид формы Form1 для проекта LISTBOX1

Листинг 25.7. Настройка свойств

```
label3: Text = Insert, TextAlign = MiddleCenter,
  AutoSize = False, BorderStyle = FixedSingle
label4: Text = Move Up, TextAlign = MiddleCenter,
  AutoSize = False, BorderStyle = FixedSingle
label5: Text = Move Down, TextAlign = MiddleCenter,
  AutoSize = False, BorderStyle = FixedSingle
label6: Text = Save To File,
  TextAlign = MiddleCenter, AutoSize = False,
  BorderStyle = FixedSingle
label7: Text = Load From File,
  TextAlign = MiddleCenter, AutoSize = False,
  BorderStyle = FixedSingle
```

Листинг 25.8. Обработчики `label3.Click`, `label4.Click`, `label5.Click`, `label6.Click` и `label7.Click`

```
private void label3_Click(object sender, EventArgs e)
{
  int i = listBox1.SelectedIndex;
  if (i == -1)
    label1_Click(this, null);
  else
  {
    listBox1.Items.Insert(i, comboBox1.Text);
    listBox1.SelectedIndex = i;
  }
}
private void label4_Click(object sender, EventArgs e)
{
  int i = listBox1.SelectedIndex;
  if (i <= 0)
  {
    Console.Beep();
    return;
  }
  object x = listBox1.Items[i];
  listBox1.Items[i] = listBox1.Items[i - 1];
```

```
  listBox1.Items[i - 1] = x;
  listBox1.SelectedIndex = i - 1;
}
private void label5_Click(object sender, EventArgs e)
{
  int i = listBox1.SelectedIndex;
  if (i == -1 || i == listBox1.Items.Count - 1)
  {
    Console.Beep();
    return;
  }
  object x = listBox1.Items[i];
  listBox1.Items[i] = listBox1.Items[i + 1];
  listBox1.Items[i + 1] = x;
  listBox1.SelectedIndex = i + 1;
}
private void label6_Click(object sender, EventArgs e)
{
  if (listBox1.Items.Count == 0)
  {
    Console.Beep();
    return;
  }
  StreamWriter sw = new StreamWriter("LISTBOX1.dat");
  foreach (object o in listBox1.Items)
    sw.WriteLine(o);
  sw.Close();
}
private void label7_Click(object sender, EventArgs e)
{
  if (!File.Exists("LISTBOX1.dat"))
  {
    Console.Beep();
    return;
  }
  listBox1.Items.Clear();
  StreamReader sr = new StreamReader("LISTBOX1.dat");
  while (!sr.EndOfStream)
    listBox1.Items.Add(sr.ReadLine());
  sr.Close();
  listBox1.SelectedIndex = listBox1.Items.Count - 1;
}
```

Результат: щелчок мышью на метке **Insert** вставляет новый элемент перед выделенным и выделяет вставленный элемент (в случае пустого списка команда **Insert** и команда **Add** выполняют одинаковые действия). Команды **Move Up** и **Move Down** позволяют перемещать выделенный элемент соответственно вверх и вниз по списку, сохраняя его выделение (при попытке выполнить перемещение первого элемента вверх или последнего элемента вниз выдается звуковой сигнал). Команда **Save To File** позволяет сохранить содержимое *непустого* списка в файле LISTBOX1.dat, а команда **Load From File** — загрузить в список данные из этого файла (если файл отсутствует, то при попытке загрузить данные выдается звуковой сигнал).

25.4. Выполнение операций над списком с помощью мыши

Свяжите событие `DoubleClick` компонента `listBox1` с существующим обработчиком `label2_Click` (его текст приведен в листинге 25.6).

Положите свойство `AllowDrop` компонента `listBox1` равным **`True`** и добавьте в класс `Form1` поле, которое в режиме перетаскивания будет содержать индекс перетаскиваемого элемента (source) из списка `listBox1`:

```
private int iSrc;
```

Определите обработчики события `MouseDown` для компонентов `comboBox1` и `listBox1`, а также обработчики событий `DragEnter` и `DragDrop` для компонента `listBox1` (листинг 25.9).

Листинг 25.9. Обработчики `comboBox1.MouseDown`, `listBox1.MouseDown`, `listBox1.DragEnter` и `listBox1.DragDrop`

```
private void comboBox1_MouseDown(object sender, MouseEventArgs e)
{
  if (e.Button == MouseButtons.Right)
    DoDragDrop(comboBox1.Text, DragDropEffects.Copy);
}
private void listBox1_MouseDown(object sender, MouseEventArgs e)
{
  if (e.Button == MouseButtons.Right)
  {
    iSrc = listBox1.IndexFromPoint(e.Location);
    if (iSrc != ListBox.NoMatches)
```

```
      DoDragDrop(listBox1.Items[iSrc].ToString(), DragDropEffects.Move);
  }
}
private void listBox1_DragEnter(object sender, DragEventArgs e)
{
  e.Effect = DragDropEffects.All;
}
private void listBox1_DragDrop(object sender, DragEventArgs e)
{
  if (e.AllowedEffect == DragDropEffects.Move)
    listBox1.Items.RemoveAt(iSrc);
  string s = e.Data.GetData(typeof(string)) as string;
  int iTrg = listBox1.IndexFromPoint(listBox1.PointToClient(new Point(e.X, e.Y)));
  if (iTrg == ListBox.NoMatches)
    listBox1.SelectedIndex = listBox1.Items.Add(s);
  else
  {
    listBox1.Items.Insert(iTrg, s);
    listBox1.SelectedIndex = iTrg;
  }
}
```

Результат: при двойном щелчке на элементе списка происходит удаление этого элемента (благодаря связыванию события `DoubleClick` компонента `listBox1` с обработчиком `label2_Click`). Кроме того, теперь для перемещения элемента списка на новую позицию можно использовать механизм drag & drop: достаточно зацепить любой (не обязательно выделенный) элемент правой кнопкой мыши и перетащить его на новое место. Текст из выпадающего списка `comboBox1` также можно поместить в список с помощью перетаскивания правой кнопкой мыши. Если текст перетаскивается на существующий элемент списка, то он вставляется в указанную позицию, а если текст перетаскивается на свободную область списка, то он добавляется к списку. В любом случае он становится выделенным элементом.

При перетаскивании текста из выпадающего списка `comboBox1` изображение курсора содержит символ **+**, что является признаком режима **Copy** (Копировать); при перетаскивании элемента списка `listBox1` на новое место курсор не содержит символа **+**, что является признаком режима **Move** (Переместить).

Недочет: в начале перетаскивания из выпадающего списка курсор имеет вид запрещающего знака.

Исправление: положите свойство `AllowDrop` компонента `comboBox1` равным **`True`** и определите обработчик события DragEnter для компонента comboBox1 (листинг 25.10).

Листинг 25.10. Обработчик `comboBox1.DragEnter`

```
private void comboBox1_DragEnter(object sender, DragEventArgs e)
{
  e.Effect = DragDropEffects.Copy;
}
```

Результат: теперь при перетаскивании элемента из выпадающего списка курсор над данным списком является разрешающим (хотя, разумеется, отпускание элемента над выпадающим списком не приведет ни к каким результатам). Отметим, что если перетаскивание начато из обычного списка `listBox1`, то над компонентом `comboBox1` курсор будет иметь запрещающий вид (это связано с тем, что выпадающий список в качестве приемника перетаскивания реагирует только на режим **Copy**).

Глава 13. Просмотр изображений: проект IMGVIEW

13.1. Добавление в окно *Toolbox* новых компонентов

После создания проекта IMGVIEW положите свойство `Text` формы `Form1` равным **`Image Viewer`**, а ее свойство `StartPosition` равным **`CenterScreen`**.

Перед размещением на форме новых компонентов добавим в окно компонентов **Toolbox** три компонента, обеспечивающие просмотр элементов файловой системы компьютера: `DriveListBox` (просмотр и выбор дисков), `DirListBox` (просмотр и выбор каталогов, находящихся на указанном диске) и `FileListBox` (просмотр и выбор файлов, находящихся в указанном каталоге).

Для добавления указанных компонентов в окно **Toolbox** перейдите в группу компонентов, в которую следует поместить новые компоненты (например, **All Windows Forms**), вызовите ее контекстное меню, щелкнув правой кнопкой мыши, и выполните команду **Choose Items…**. Данная команда может выполняться достаточно долгое время, поскольку в ходе ее выполнения формируется список всех компонентов, доступных в .NET Framework. После завершения формирования списка доступных компонентов он отображается на экране в окне **Choose Toolbox Items**. На вкладке **.NET Framework Components** этого окна установите флажки рядом с именами компонентов `DriveListBox`, `DirListBox` и `FileListBox`, после чего нажмите клавишу <Enter> или кнопку **OK**.

Результат: отмеченные компоненты добавлены в текущую группу в окне **Toolbox**. С каждым из добавленных компонентов связывается одинаковое изображение в виде шестеренки:  .

13.2. Просмотр изображений из файлов на текущем диске

Разместите в форме `Form1` компонент типа `SplitContainer` (будем называть его *split-контейнером*). Этот компонент получит по умолчанию имя `splitContainer1` и сразу займет всю свободную часть формы (поскольку его свойство `Dock` по умолчанию равно **`Fill`**). Компонент `splitContainer1` состоит из двух дочерних панелей (левой с именем `Panel1` и правой с именем `Panel2`); между этими панелями находится *разделитель* (splitter) — область, зацепив за которую можно увеличить ширину одной панели за счет уменьшения ширины другой. Каждую из дочерних панелей можно выделить; при этом в окне **Properties** отображаются свойства выделенной панели. Разделитель выделить нельзя; при щелчке на нем мышью выделяется весь компонент `splitContainer1` (заметим, что это наиболее удобный способ выделения split-контейнера, так как остальная его часть занята дочерними панелями).

Разместите на панели `Panel2` компонента `splitContainer1` еще один split-контейнер (он получит имя `splitContainer2`).

Хотя второй split-контейнер помещен на одну из панелей первого split-контейнера, визуально форма будет содержать три равноправные панели одинаковой высоты, разделенные одинаковыми по ширине разделителями. Определить, к какому контейнеру принадлежит та или иная панель, можно с помощью верхней части окна **Properties**. При последовательном выделении панелей формы слева направо в верхней части окна **Properties** будут указаны следующие имена: `splitContainer1.Panel1`, `splitContainer2.Panel1` и `splitContainer2.Panel2`. Заметим, что щелчок мышью на разделителе, расположенном между двумя левыми панелями, приведет к выделению компонента `splitContainer1`, а щелчок на разделителе между двумя правыми панелями — к выделению компонента `splitContainer2`.

Теперь заполним панели split-контейнеров реальным содержимым. На панель `Panel1` компонента `splitContainer1` поместите компонент типа `DirListBox`, на панель `Panel1` компонента `splitContainer2` поместите компонент типа `FileListBox`, на панель `Panel2` компонента `splitContainer2` поместите компонент типа `PictureBox` (добавленные компоненты получат имена `dirListBox1`, `fileListBox1` и `pictureBox1`). Настройте свойства добавленных компонентов (листинг 13.1; рис. 13.1). Напомним, что при настройке свойства `Dock` в окне **Properties** отображается специальная панель; для выбора в этой панели требуемого варианта **`Fill`** надо щелкнуть мышью на центральном прямоугольнике.

Определите обработчики события `Change` для компонента `dirListBox1` и события `SelectedIndexChanged` для компонента `fileListBox1` (листинг 13.2).

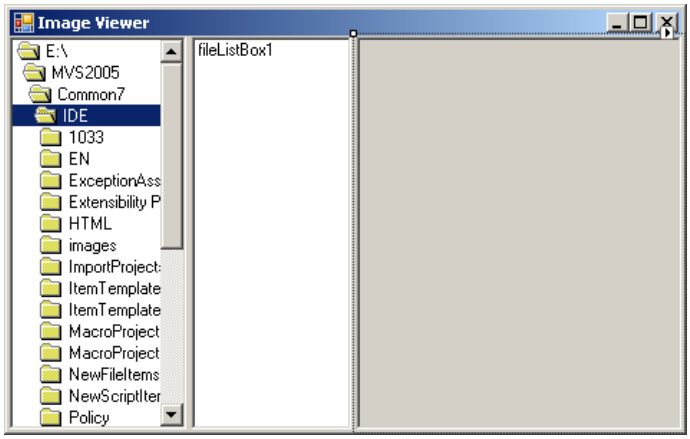

Рис. 13.1. Вид формы Form1 для проекта IMGVIEW на начальном этапе разработки

Листинг 13.1. Настройка свойств

```
dirListBox1.Dock = Fill
fileListBox1: Dock = Fill, IntegralHeight = False,
 Pattern = *.bmp;*.jpg;*.png;*.gif;*.ico;*.wmf;*.emf
pictureBox1: Dock = Fill, SizeMode = Zoom,
  BorderStyle = Fixed3D
```

Листинг 13.2. Обработчики `dirListBox1.Change` и `fileListBox1.SelectedIndexChanged`

```
private void dirListBox1_Change(object sender, EventArgs e)
{
  fileListBox1.Path = dirListBox1.Path;
}
private void fileListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  try
  {
    pictureBox1.Image = new Bitmap(fileListBox1.Path + "\\" + fileListBox1.FileName);
  }
  catch
  {
    pictureBox1.Image = null;
  }
```

```
}
```

Результат: программа позволяет перемещаться по каталогам текущего диска. При запуске программы выбирается ее *рабочий каталог*; по умолчанию этим каталогом считается каталог, из которого запущен exe-файл (в нашем случае рабочим каталогом будет подкаталог bin\Debug каталога с проектом IMGVIEW, поэтому для ускорения тестирования проекта IMGVIEW в его подкаталог bin\Debug целесообразно скопировать несколько графических файлов разных типов).

Для выбора нового каталога требуется выполнить двойной щелчок мышью на его имени в списке каталогов; при этом в списке каталогов дополнительно отображаются имена всех подкаталогов выбранного каталога, а в списке файлов выводятся имена графических файлов (с расширениями bmp, jpg, png, gif, ico, wmf, emf), находящихся в выбранном каталоге. При щелчке на имени файла (или при перемещении по списку файлов с помощью клавиш со стрелками) на правой панели в компоненте `pictureBox1` выводится изображение из выбранного файла; при этом изображение масштабируется по размерам панели с сохранением его исходных пропорций.

Ширину панелей можно изменять путем перетаскивания мышью разделителей, расположенных между ними (при перемещении курсора мыши на эти разделители он принимает вид двунаправленной стрелки). При изменении ширины формы происходит пропорциональное изменение ширины каждой из трех панелей.

Недочет 1: при изменении размеров формы (и происходящем при этом синхронном изменении ширины ее панелей) содержимое списка каталогов и списка файлов неприятно мерцает в результате многократной перерисовки.

Исправление: для компонентов `splitContainer1` и `splitContainer2` положите их свойство `FixedPanel` равным **`Panel1`.**

Результат: теперь изменение ширины формы сказывается только на правой панели, содержащей изображение; ширина панелей со списками каталогов и файлов не изменяется. В подобной ситуации мерцание списков практически исчезает (мерцает лишь выделенный элемент в списке каталогов).

Недочет 2: при смене каталога на правой панели сохраняется ранее загруженное изображение.

Исправление: добавьте новые операторы в конструктор класса `Form1` и метод `dirListBox1_Change` (листинг 13.3).

Листинг 13.3. Новые варианты конструктора класса `Form1` и метода `dirListBox1_Change`

```
public Form1()
{
  InitializeComponent();
  dirListBox1_Change(this, null);
}
private void dirListBox1_Change(object sender, EventArgs e)
{
  fileListBox1.Path = dirListBox1.Path;
  if (fileListBox1.Items.Count > 0)
    fileListBox1.SelectedIndex = 0;
  else
    pictureBox1.Image = null;
}
```

Результат: теперь при смене каталога в списке файлов автоматически выбирается первый элемент, и его изображение выводится на правой панели. Если каталог не содержит графических файлов, то правая панель очищается.

Примечание

Поскольку при запуске программы событие `Change` компонента `dirListBox1` автоматически не возникает, приходится имитировать его вызов, запуская обработчик `dirListBox1_Change` при создании формы (после создания и инициализации всех ее компонентов).

Недочет 3: в списке каталогов нельзя сменить каталог с помощью клавиатуры; можно лишь перемещаться по списку отображаемых каталогов.

Исправление: определите обработчик события `KeyPress` для компонента `dirListBox1` (листинг 13.4).

Листинг 13.4. Обработчик `dirListBox1.KeyPress`

```
private void dirListBox1_KeyPress(object sender, KeyPressEventArgs e)
{
  if (e.KeyChar == '\r')
    dirListBox1.Path = dirListBox1.get_DirList(dirListBox1.DirListIndex);
}
```

Результат: теперь для смены каталога достаточно выделить его в списке каталогов и нажать клавишу <Enter>.

13.3. Стыковка компонентов и ее особенности

В этом разделе мы разместим в нижней части формы вспомогательную панель. Поскольку вся форма заполнена дочерними панелями split-контейнеров, щелчок в любой точке формы при размещении в ней нового компонента приведет к тому, что он разместится на одной из этих дочерних панелей. Однако имеется простая возможность разместить компонент непосредственно на форме, даже если вся ее клиентская область занята другими компонентами-контейнерами: для этого достаточно при помещении компонента на форме *щелкнуть на ее заголовке*.

Выполним это действие для размещения в форме нового компонента типа `Panel`. Добавленная панель получит имя `panel1`. Настройте ее свойство `Dock`, положив его равным **`Bottom`** (для этого во вспомогательной панели окна **Properties**, связанной с этим свойством, надо щелкнуть мышью на нижнем прямоугольнике).

Ошибка: новая панель размещается в нижней части формы (*"пристыковывается"* к нижней границе), однако располагается не рядом, а поверх остальных панелей, закрывая их нижнюю часть.

Исправление: не снимая выделения с новой панели `panel1`, нажмите на кнопку **Send to Back**  (вторая справа кнопка на панели выравнивания **Layout**).

Результат: теперь компонент `splitContainer1`, свойство `Dock` которого имеет значение **`Fill`**, занимает всю область формы, за исключением той нижней части, которая отведена для панели `panel1` (рис. 13.2).

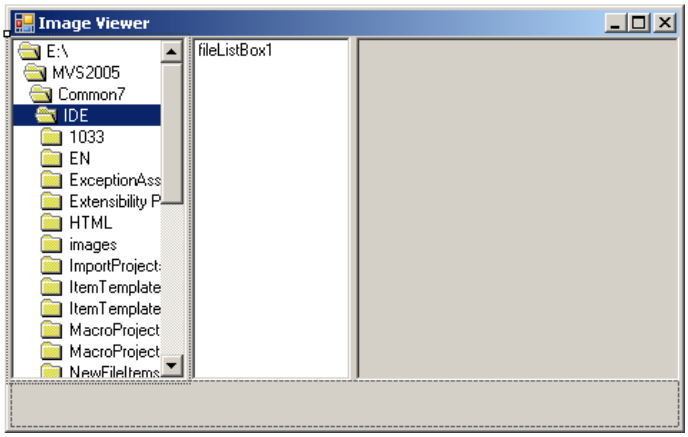

Рис. 13.2. Вид формы Form1 для проекта IMGVIEW на промежуточном этапе разработки

13.4. Возможность смены диска

Разместите на панели `panel1` компонент `DriveListBox` (это один из трех компонентов, добавленных нами в окно **Toolbox** в разд. 13.1); его положение настройте в соответствии с рис. 13.3, приведенном в следующем разделе. Новый компонент получит имя `driveListBox1`; определите для него обработчик события `SelectedIndexChanged` (листинг 13.5).

Листинг 13.5. Обработчик `driveListBox1.SelectedIndexChanged`

```
private void driveListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  dirListBox1.Path = driveListBox1.Drive[0] + ":\\";
}
```

Результат: используя добавленный выпадающий список `driveListBox1`, можно выбирать любой доступный диск; при этом автоматически обновляются списки каталогов и файлов, а выбранным становится корневой каталог выбранного диска.

Недочет: если выбранный дисковод (или привод для компакт-дисков) не содержит носителя, то возникает исключительная ситуация типа `IOException` с сообщением "Device unavailable". При этом, если продолжить выполнение программы, то в выпадающем списке останется выбранным недоступный дисковод.

Исправление: измените метод `driveListBox1_SelectedIndexChanged`, добавив в его начало новые операторы (листинг 13.6).

Листинг 13.6. Новый вариант метода `driveListBox1_SelectedIndexChanged`

```
private void driveListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  while (!new System.IO.DriveInfo(driveListBox1.Drive).IsReady)
    if (MessageBox.Show("Drive " + driveListBox1.Drive +
        " is not ready.", "Warning", MessageBoxButtons.RetryCancel,
        MessageBoxIcon.Warning) != DialogResult.Retry)
    {
      driveListBox1.SelectedIndexChanged -=
        driveListBox1_SelectedIndexChanged;
      driveListBox1.Drive = dirListBox1.Path[0] + ":";
      driveListBox1.SelectedIndexChanged +=
        driveListBox1_SelectedIndexChanged;
      return;
    }
  dirListBox1.Path = driveListBox1.Drive[0] + ":\\";
}
```

Результат: при попытке выбрать недоступный диск появляется окно с сообщением "Drive *<имя диска>* is not ready" и предлагаются два варианта продолжения: **Повтор** — повторить попытку чтения диска, **Отмена** — отменить выбор диска.

При отмене выбора происходит возврат к ранее выбранному диску, причем выбранный каталог на этом диске не изменяется.

Примечание

При восстановлении ранее выбранного диска приходится временно отключать обработчик `driveListBox1_SelectedIndexChanged` от события `SelectedIndexChanged`. Если этого не делать, то присваивание свойству `Drive` первого символа строки `dirListBox1.Path` приведет к повторному запуску обработчика, который в этом ситуации проработает успешно и, следовательно, установит для восстановленного диска не ранее выбранный, а корневой каталог. По поводу явного отключения и подключения обработчиков см. также разд. 4.2–4.3.

13.5. Настройка режима просмотра изображений

Разместите на панели `panel1` флажок `checkBox1`, настройте его свойства (листинг 13.7, рис. 13.3) и определите для него обработчик события `CheckStateChanged` (листинг 13.8).

Листинг 13.7. Настройка свойств

```
checkBox1: Text = &Zoom Image, ThreeState = True,
  CheckState = Checked
```

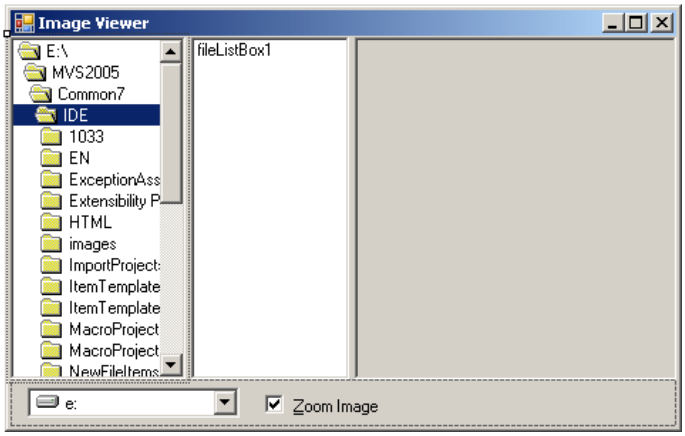

Рис. 13.3. Окончательный вид формы Form1 для проекта IMGVIEW

Листинг 13.8. Обработчик `checkBox1.CheckStateChanged`

```
private void checkBox1_CheckStateChanged(object sender, EventArgs e)
{
  switch (checkBox1.CheckState)
  {
    case CheckState.Checked:
      pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;
      break;
    case CheckState.Indeterminate:
      pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
      break;
    case CheckState.Unchecked:
      pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;
      break;
  }
}
```

Результат: флажок **Zoom Image** может принимать три состояния, определяя режим масштабирования изображения: при установленном флажке ("галочка" на белом фоне) изображение, как и раньше, масштабируется с сохранением пропорций; при затененном флажке ("галочка" на сером фоне) изображение растягивается по всей правой панели, при этом пропорции рисунка *не сохраняются*; наконец, при сброшенном флажке изображение выводится без масштабирования и размещается в центре правой панели. Заметим, что отключать масштабирование целесообразно при просмотре значков и небольших рисунков, поскольку при их сильном увеличении изображение получается размытым.

При щелчке мышью на флажке или нажатии комбинации клавиш <Alt>+<Z> состояние флажка меняется циклически в такой последовательности: установленный — затененный — отключенный.

13.6. Сохранение в реестре Windows информации о состоянии программы

В начало файла Form1.cs добавьте оператор

```
using Microsoft.Win32;
```

В описание класса `Form1` добавьте описание нового поля:

```
private string regKeyName = "Software\\CSExamples\\IMGVIEW";
```

Определите обработчик события `FormClosed` для формы `Form1` (листинг 13.9).

Листинг 13.9. Обработчик `Form1.FormClosed`

```
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
  RegistryKey rk = null;
  try
  {
    rk = Registry.CurrentUser.CreateSubKey(regKeyName);
    if (rk == null)
      return;
    rk.SetValue("FormWidth", Width);
    rk.SetValue("FormHeight", Height);
    rk.SetValue("Split1", splitContainer1.SplitterDistance);
    rk.SetValue("Split2", splitContainer2.SplitterDistance);
    rk.SetValue("Zoom", (int)checkBox1.CheckState);
    rk.SetValue("Path", dirListBox1.Path);
    rk.SetValue("File", fileListBox1.FileName);
  }
  finally
  {
    if (rk != null)
      rk.Close();
  }
}
```

Результат: теперь при завершении работы программы сведения о ее текущем состоянии записываются в реестр Windows. Для того чтобы в этом убедиться, следует запустить программу **regedit** (Редактор реестра). Проще всего запустить данную программу, выполнив в главном меню Windows команду **Пуск | Выполнить…** и введя в появившееся окно имя программы: **regedit**. В редакторе реестра надо выбрать раздел HKEY_CURRENT_USER, а в нем подраздел Software\CSExamples\IMGVIEW. В результате на правой панели редактора реестра будут отображены поля выбранного подраздела и их значения. Подраздел должен содержать (помимо поля **По умолчанию**, которое не будет использоваться в нашей программе) семь полей: **File**, **FormHeight**, **FormWidth**, **Path**, **Split1**, **Split2**, **Zoom** (поля **File** и **Path** являются строковыми, остальные — целочисленными). В разд. 13.7 мы добавим в программу фрагмент, позволяющий считывать из реестра эти данные.

13.7. Восстановление из реестра Windows информации о состоянии программы

Измените конструктора класса `Form1` (листинг 13.10).

Листинг 13.10. Новый вариант конструктора класса `Form1`

```
public Form1()
{
  InitializeComponent();
  RegistryKey rk = null;
  string s = "";
  try
  {
    rk = Registry.CurrentUser.OpenSubKey(regKeyName);
    if (rk != null)
    {
      Width = (int)rk.GetValue("FormWidth", Width);
      Height = (int)rk.GetValue("FormHeight", Height);
      splitContainer1.SplitterDistance = (int)rk.GetValue("Split1",
        splitContainer1.SplitterDistance);
      splitContainer2.SplitterDistance = (int)rk.GetValue("Split2",
        splitContainer2.SplitterDistance);
      checkBox1.CheckState = (CheckState)rk.GetValue("Zoom",
        checkBox1.CheckState);
      s = (string)rk.GetValue("Path", "");
      if (System.IO.Directory.Exists(s))
      {
        driveListBox1.Drive = s[0] + ":";
        dirListBox1.Path = s;
      }
      s = (string)rk.GetValue("File", "");
    }
  }
  finally
  {
    if (rk != null)
      rk.Close();
  }
  fileListBox1.SelectedItem = s;
}
```

Результат: при запуске программы данные из реестра Windows используются для восстановления ее состояния. Если данные в реестре отсутствуют, то устанавливаются настройки по умолчанию. Поскольку с момента последнего сохранения данных в реестре состояние дисковой системы могло измениться, при считывании поля **Path** проверяется наличие указанного в нем каталога. Новые настройки для компонентов `driveListBox1` и `dirListBox1` устанавливаются только в случае обнаружения каталога **Path**. Проверять наличие в каталоге файла с именем, определяемым полем **File**, не требуется, так как отсутствие этого файла приведет лишь к тому, что в списке `fileListBox1` будет отсутствовать выделенный элемент.

Приведем изображение работающей программы (рис. 13.4).

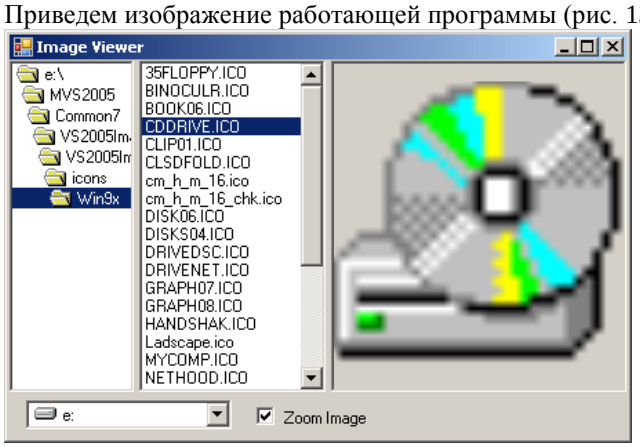

Рис. 13.4. Вид работающего приложения IMGVIEW

Глава 27. MDI-приложение: проект JPEGVIEW

27.1. Открытие и закрытие дочерних форм в MDI-приложении

После создания проекта JPEGVIEW разместите в форме `Form1` невизуальные компоненты типа `MenuStrip` и `OpenFileDialog` (эти компоненты получат имена `menuStrip1` и `openFileDialog1`; они будут размещены в области невизуальных компонентов под изображением формы, кроме того, в верхней части формы будет указано меню, связанное с компонентом `menuStrip1`).

Используя конструктор меню (см. разд. 18.1), создайте в меню `menuStrip1` пункт меню первого уровня с текстом **`&File`** и с помощью окна **Properties** измените имя этого пункта (то есть свойство `Name`) на **`file1`**. В выпадающем меню, связанном с пунктом **File**, создайте пункт меню с текстом **`&Open`**, затем пункт с текстом – (дефис; этот пункт будет преобразован в горизонтальную разделительную линию), затем пункт с текстом **`E&xit`**. Настройте свойства формы `Form1` и добавленных в нее компонентов и пунктов меню (листинг 27.1; см. также рис. 27.1).

Добавьте к проекту новую форму (она получит имя `Form2`) и разместите в ней компонент-меню `menuStrip1` и компонент-контейнер для изображений `pictureBox1`. В меню `menuStrip1` формы `Form2` также создайте пункт меню первого уровня с текстом **`&File`** и именем **`file1`**. В выпадающем меню, связанном с пунктом **File**, создайте *два* пункта меню с текстом **`&Open`** и **`&Close`**, затем пункт с текстом – (дефис), затем пункт с текстом **`E&xit`**. Настройте свойства формы `Form2` и добавленных в нее компонентов и пунктов меню (листинг 27.2; см. также рис. 27.2).

Определите обработчики события `Click` для пунктов меню `open1` и `exit1` формы `Form1` (листинг 27.3).

Определите обработчики события `Click` для пунктов меню `open1`, `close1` и `exit1` формы `Form2` (листинг 27.4).

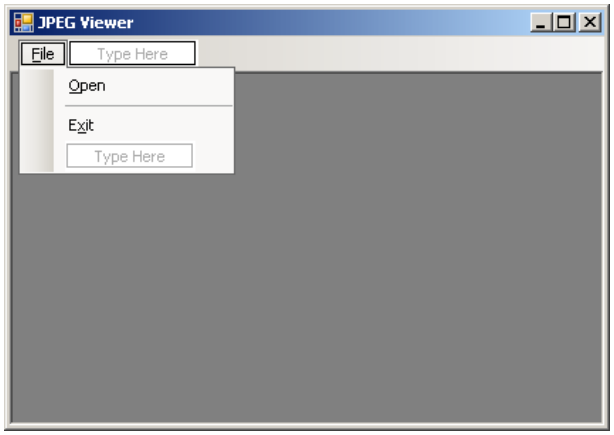

Рис. 27.1. Вид формы Form1 для проекта JPEGVIEW на начальном этапе разработки

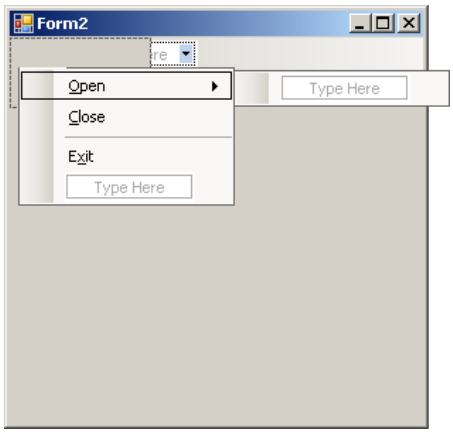

Рис. 27.2. Вид формы Form2 для проекта JPEGVIEW на начальном этапе разработки

Листинг 27.1. Настройка свойств формы `Form1` и ее компонентов

```
Form1: Text = JPEG View, IsMdiContainer = True,
  StartPosition = CenterScreen
openFileDialog1: Title = Открытие файла,
  Filter = JPEG Images|*.jpg; *.jpeg
пункт меню Open (группа File): Name = open1
пункт меню Exit (группа File): Name = exit1
```

Листинг 27.2. Настройка свойств формы `Form2` и ее компонентов

```
Form2: Text = пустая строка, ShowInTaskbar = False,
  AutoScroll = True
pictureBox1: SizeMode = AutoSize, Location = 0; 0,
  Modifiers = Internal
file1: MergeAction = Replace
пункт меню Open (группа File): Name = open1
пункт меню Close (группа File): Name = close1
пункт меню Exit (группа File): Name = exit1
```

Листинг 27.3. Обработчики `open1.Click` и `exit1.Click` (для пунктов меню формы `Form1`)

```
internal void open1_Click(object sender, EventArgs e)
// модификатор доступа изменен на internal
{
  openFileDialog1.FileName = "";
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
    // создание child-формы и ее настройка
    Form2 f = new Form2();
    string n = openFileDialog1.FileName;
    Image i = Image.FromFile(n);
    f.pictureBox1.Image = i;
    f.Text = string.Format("{0} ({1}x{2})",
      System.IO.Path. GetFileNameWithoutExtension(n), i.Width, i.Height);
    f.MdiParent = this;
    f.Show();
  }
}
private void exit1_Click(object sender, EventArgs e)
{
  Close();
}
```

Листинг 27.4. Обработчики `open1.Click`, `close1.Click` и `exit1.Click` (для пунктов меню формы `Form2`)

```
private void open1_Click(object sender, EventArgs e)
{
  (MdiParent as Form1).open1_Click(this, null);
}
private void close1_Click(object sender, EventArgs e)
{
  // закрытие дочерней формы:
  Close();
}
private void exit1_Click(object sender, EventArgs e)
{
  // закрытие главной формы:
  MdiParent.Close();
```

```
}
```

Результат: из главной формы **JPEG View** с помощью команды **Open** можно загружать графические файлы в формате JPEG (jpg-файлы). Каждый из загруженных файлов отображается в отдельном окне (*дочерней форме*, child-форме), причем в заголовке child-формы указывается имя файла и размер изображения в пикселах (например, **200x350**). Начальный размер и положение child-формы определяются по умолчанию (так как ее свойство `StartPosition` равно **`WindowsDefaultLocation`**). Если размеры child-формы недостаточны для отображения всего изображения, то в ней появляются полосы прокрутки. При разворачивании child-формы ее заголовок совмещается с заголовком главной формы. При сворачивании child-формы ее заголовок помещается в нижней части главной формы. Если открыта хотя бы одна child-форма, то в меню главной формы появляется дополнительная команда **Close**, которая позволяет закрыть активную child-форму. Если закрыты все child-формы, то восстанавливается исходное меню главной формы.

Для закрытия активной child-формы можно использовать комбинацию <Ctrl>+<F4>. Кроме того, предусмотрены комбинации для активизации следующей и предыдущей child-формы: это, соответственно, <Ctrl>+<F6> и <Ctrl>+<Shift>+<F6>. Переключаться между открытыми child-формами можно также с помощью клавиш со стрелками. Обработка всех упомянутых клавиш производится в MDI-приложении автоматически.

Недочет: если ширина дочерней формы превосходит размеры рисунка, то на верхней части дочерней формы видна полоса меню.

Исправление: для компонента `menuStrip1` формы `Form2` положите значение свойства `Visible` равным **`False`**.

Результат: теперь меню на дочерних формах не отображается, причем это не влияет на вид пунктов, добавленных из этого меню в меню главной формы.

27.2. Стандартные действия над дочерними формами

Создайте в меню `menuStrip1` формы `Form1` новый пункт меню первого уровня с текстом **`&Window`** и с помощью окна **Properties** измените имя этого пункта (то есть свойство `Name`) на **`window1`**. В выпадающем меню, связанном с пунктом **Window**, создайте пункты меню с текстом **`&HTile`**, **`&VTile`**, **`&Cascade`**, **`&Arrange Icons`** и настройте свойства `Name` этих пунктов, положив их равными **`hTile1`**, **`vTile2`**, **`cascade1`**, **`arrangeIcons1`** соответственно (см. рис. 27.3).

Добавьте новые операторы в конструктор класса `Form1` (листинг 27.5). Определите обработчик события `Click` для пункта меню `hTile1` (листинг 27.6), после чего свяжите созданный обработчик с событием `Click` пунктов меню `vTile2`, `cascade1` и `arrangeIcons1`.

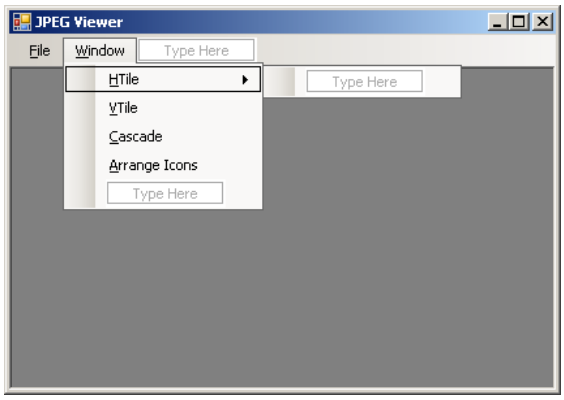

Рис. 27.3. Вид формы Form1 для проекта JPEGVIEW на промежуточном этапе разработки

Листинг 27.5. Новый вариант конструктора класса `Form1`

```
public Form1()
{
  InitializeComponent();
  hTile1.Tag = MdiLayout.TileHorizontal;
  vTile1.Tag = MdiLayout.TileVertical;
  cascade1.Tag = MdiLayout.Cascade;
  arrangeIcons1.Tag = MdiLayout.ArrangeIcons;
}
```

Листинг 27.6. Обработчик `hTile1.Click`

```
private void hTile1_Click(object sender, EventArgs e)
{
  LayoutMdi((MdiLayout)(sender as ToolStripMenuItem).Tag);
}
```

Результат: с помощью команд меню группы **Window** можно выполнять стандартные действия над child-формами.

- **HTile** и **VTile**: размещение всех child-форм без перекрытий (*черепицей*) так, чтобы они покрывали всю главную форму. Данные команды различаются способом расположения форм: **HTile** размещает формы сверху вниз (растягивая их по горизонтали), а **VTile** — слева направо (растягивая их по вертикали). Иногда эти команды действуют одинаково (например, для четырех или пяти child-форм).

- **Cascade**: размещение всех child-форм *каскадом* так, чтобы заголовки всех форм были видны на экране. При этом устанавливается единый стандартный размер для всех child-форм, зависящий от текущего размера главной формы.
- **Arrange Icons**: размещение всех *свернутых* child-форм на нижней границе главной формы.

Недочет: при отсутствии child-форм команды группы **Window** теряют смысл, однако по-прежнему доступны в меню.

Исправление: для пункта меню `window1` формы `Form1` установите значение свойства `Modifiers` равным **`Internal`**, а значение свойства `Visible` равным **`False`**.

В методе `open1_Click` формы `Form1` (листинг 27.3) перед оператором
```
f.Show();
```
вставьте
```
window1.Visible = true;
```
Определите обработчик `FormClosed` формы `Form2` (листинг 27.7).

Листинг 27.7. Обработчик `Form2.FormClosed`
```
private void Form2_FormClosed(object sender, FormClosedEventArgs e)
{
  if (MdiParent.MdiChildren.Length == 1)
  // осталась единственная child-форма,
  // которая в данный момент закрывается
    (MdiParent as Form1).window1.Visible = false;
}
```
Результат: пункт меню **Window** отображается только при наличии хотя бы одной открытой child-формы.

27.3. Добавление в меню списка открытых дочерних форм

Для компонента `menuStrip1` формы `Form1` установите значение свойства `MdiWindowListItem` равным **`window1`**.

Результат: в группе команд **Window** теперь выводится список всех открытых child-форм; около активной child-формы отображается "галочка". Щелкнув на элементе из этого списка, можно активизировать соответствующую child-форму. Для выбора элемента из списка можно также нажать клавишу, соответствующую его порядковому номеру (от <1> до <9>). Если открыто более 9 окон, то ниже пункта, соответствующего девятому окну, выводится пункт **More Windows…**, выбрав который, можно отобразить на экране вспомогательное диалоговое окно **Select Window** со списком всех дочерних окон.

27.4. Одновременное закрытие всех дочерних форм

Добавьте в меню **File** формы `Form2` новый пункт с текстом **`Close &All`**. Данный пункт надо добавить после пункта **Close**; для этого надо выделить разделительную черту, расположенную ниже команды **Close**, нажать правую кнопку мыши и из появившегося меню выбрать команду **Insert | MenuItem**. Можно также поступить по-другому: создать пункт **Close All** в конце меню **File**, а затем перетащить его мышью на новое место. Положите свойство `Name` созданного пункта меню равным **`closeAll1`** и определите обработчик события `Click` для этого пункта меню (листинг 27.8).

Листинг 27.8. Обработчик `closeAll1.Click`
```
private void closeAll1_Click(object sender, EventArgs e)
{
  foreach (Form f in MdiParent.MdiChildren)
    f.Close();
}
```
Результат: при выполнении команды **Close All** происходит закрытие всех child-форм.

Примечание
Команда **Close All** включена в меню формы `Form2`, так как она должна быть доступной только при наличии в приложении открытых child-форм.

Комментарий

В методе `closeAll1_Click` использовано свойство-массив `MdiChildren` класса `Form`, содержащее все открытые в данный момент child-формы. Для перебора всех дочерних форм в данном случае удобно использовать цикл `foreach`. Заметим, что цикл вида
```
for (int i = 0; i < MdiParent.MdiChildren.Length; i++)
  MdiParent.MdiChildren[i].Close();
```
который выполняет, казалось бы, то же самое действие, будет работать неправильно или даже приведет к ошибке времени выполнения. Это связано с тем, что при удалении очередной child-формы массив `MdiChildren` изменяется, а его размер уменьшается, поэтому цикл может завершиться еще до удаления всех child-форм. Однако следующий вариант цикла
```
for (int i = MdiParent.MdiChildren.Length - 1; i >= 0; i++)
  MdiParent.MdiChildren[i].Close();
```
тоже работает неправильно! В этом случае причиной ошибки является то обстоятельство, что после разрушения child-формы, *из которой вызван данный обработчик*, ее свойство `MdiParent` станет недействительным, и попытка обращения к нему приведет к ошибке времени выполнения. Правильно будет работать вариант цикла, в котором используется вспомогательная переменная для хранения значения свойства `MdiParent`:

```
Form f = MdiParent;
for (int i = f.MdiChildren.Length - 1; i >= 0; i++)
  f.MdiChildren[i].Close();
```

Осталось объяснить, почему правильно работает вариант с циклом `foreach`. Перед выполнением этого цикла ссылка на массив `MdiParent.MdiChildren` сохраняется во вспомогательной переменной, к которой и выполняется обращение на каждой итерации цикла. Изменение свойства `MdiChildren` в этой ситуации не может нарушить нормальную работу цикла `foreach`. Действительно, при изменении состава child-форм создается *новый массив* с оставшимися child-формами, ссылка на который записывается в свойство `MdiChildren`, стирая предыдущую ссылку, однако подобные действия не могут затронуть "старый" массив child-форм, ссылка на который хранится во временной переменной, созданной при запуске цикла `foreach`.

27.5. Масштабирование изображения

Добавьте в меню **File** формы `Form2` новый пункт с текстом **`&Stretch`**. Данный пункт надо добавить после пункта **Close All**, используя один из способов, описанных в начале разд. 27.4. Положите свойство `Name` созданного пункта меню равным **`stretch1`**, свойство `CheckOnClick` равным **`True`** и определите обработчик события `CheckedChanged` для этого пункта меню (листинг 27.9).

Листинг 27.9. Обработчик `stretch1.CheckedChanged`

```
private void stretch1_CheckedChanged(object sender, EventArgs e)
{
  if (stretch1.Checked)
  {
    pictureBox1.Dock = DockStyle.Fill;
    pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;
  }
  else
  {
    pictureBox1.Dock = DockStyle.None;
    pictureBox1.SizeMode = PictureBoxSizeMode.AutoSize;
  }
}
```

Результат: изображение можно масштабировать в соответствии с текущими размерами содержащей его child-формы (при этом сохраняются пропорции изображения). Для включения/выключения режима масштабирования предусмотрена команда-переключатель **Stretch**. Если режим масштабирования включен, то около этой команды в меню изображается "галочка". При отключенном режиме масштабирования устанавливается исходный размер изображения; на child-форме в этом случае могут появиться полосы прокрутки. В каждой child-форме режим масштабирования устанавливается независимо; состояние команды-переключателя **Stretch** соответствует режиму для активной в данный момент child-формы.

27.6. Автоматическая корректировка размера дочерних форм

Добавьте в меню **File** формы `Form2` два новых пункта с текстом **`&Resize`** и **`R&esize All`**. Новые пункты надо добавить после пункта **Stretch**, используя один из способов, описанных в начале разд. 27.4. Положите свойства `Name` созданных пунктов меню равными **`resize1`** и **`resizeAll1`** соответственно. Окончательный вид меню группы **File** для формы `Form2` приводится на рис. 27.4.

Определите обработчики события `Click` для новых пунктов меню (листинг 27.10).

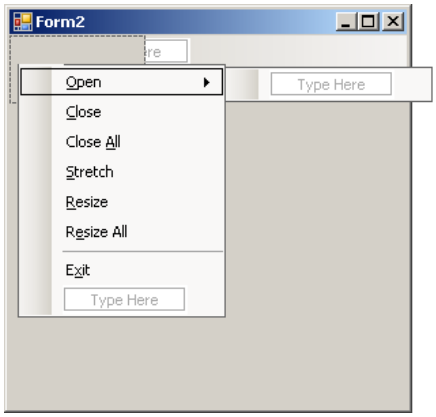

Рис. 27.4. Окончательный вид формы Form2 для проекта JPEGVIEW

Листинг 27.10. Обработчики `resize1.Click` и `resizeAll1.Click`

```
private void resize1_Click(object sender, EventArgs e)
{
  Size sz = ClientSize;
  if (stretch1.Checked)
```

```
  {
    int iw = pictureBox1.Image.Width,
      ih = pictureBox1.Image.Height,
      cw = ClientSize.Width,
      ch = ClientSize.Height;
    double x = cw / (double)iw,
      y = ch / (double)ih;
    if (x < y)
      sz.Height = (int)(x * ih);
    else
      sz.Width = (int)(y * iw);
  }
  else
    sz = pictureBox1.Size;
  ClientSize = sz;
}
private void resizeAll1_Click(object sender, EventArgs e)
{
  MdiParent.LayoutMdi(MdiLayout.Cascade);
  foreach (Form f in MdiParent.MdiChildren)
    (f as Form2).resize1_Click(f, null);
}
```

Результат: команда **Resize** настраивает размеры активной child-формы в соответствии с текущими размерами содержащегося в ней изображения. Команда **Resize All** обеспечивает настройку размеров всех child-форм; при этом все child-формы размещаются каскадом.

Примечание

Вызов метода `LayoutMdi`, обеспечивающего размещение дочерних форм каскадом, следует выполнять до настройки размеров, так как метод `LayoutMdi` сам изменяет размеры child-форм.

27.7. Дополнительные средства управления дочерними формами

Разместите в форме `Form1` *панель инструментов* — компонент типа `ToolStrip` (этот компонент получит имя `toolStrip1`). Панель инструментов автоматически пристыкуется к верхней границе формы и будет располагаться ниже главного меню `menuStrip1`.

Действуя таким же образом, как в разд. 21.1, добавьте на панель инструментов `toolStrip1` четыре компонента-кнопки типа `ToolStripButton` и измените их имена (то есть свойства `Name`) на следующие: **`open2`**, **`close2`**, **`resize2`**, **`stretch2`**, а свойства `Text` положите равными **`Open`**, **`Close`**, **`Resize`** и **`Stretch`** соответственно. Для всех четырех кнопок установите значение свойства `DisplayStyle` равным **`Text`**, значение `AutoSize` равным **`False`** и значение `Size.Width` равным **`50`** (последние две настройки необходимы для установки одинаковой ширины кнопок, так как по умолчанию ширина кнопок определяется размерами их надписи). Вид полученной панели инструментов приведен на рис. 27.5.

Измените модификаторы доступа для некоторых членов класса `Form2`: для пункта меню `stretch1` положите значение свойства `Modifiers` равным **`Internal`**, для методов `stretch1_CheckedChanged` (листинг 27.9) и `resize1_Click` (листинг 27.9) замените в их заголовках модификатор `private` на `internal`.

Вернитесь на форму `Form1` и свяжите событие `Click` кнопки `open2` с существующим обработчиком `open1_Click`, а также определите обработчики события `Click` для кнопок `close2`, `resize2` и `stretch2` (листинг 27.11).

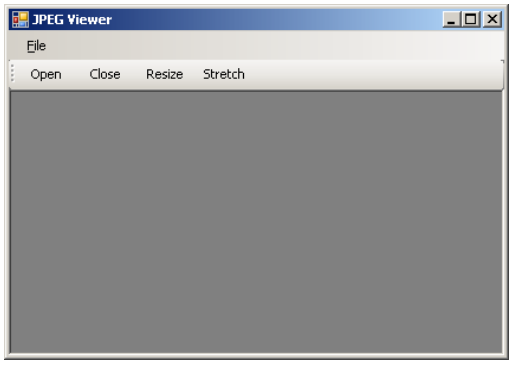

Рис. 27.5. Окончательный вид формы Form1 для проекта JPEGVIEW

Листинг 27.11. Обработчики `close2.Click`, `resize2.Click` и `stretch2.Click`

```
private void close2_Click(object sender, EventArgs e)
{
  ActiveMdiChild.Close();
}
private void resize2_Click(object sender, EventArgs e)
{
  (ActiveMdiChild as Form2).resize1_Click(this, null);
```

```
}
private void stretch2_Click(object sender, EventArgs e)
{
  ToolStripMenuItem mi = (ActiveMdiChild as Form2).stretch1;
  mi.Checked = !mi.Checked;
}
```

Результат: кнопка быстрого доступа **Open** дублирует одноименную команду меню. Кнопки **Close** и **Resize** обеспечивают закрытие и настройку размера активной child-формы, а кнопка-переключатель **Stretch** — включение и выключение масштабирования изображения в активной child-форме.

Примечание

При реализации действий по нажатию кнопок **Close**, **Resize** и **Stretch** используется свойство `ActiveMdiChild` главной формы, возвращающее активную child-форму.

Ошибка: нажатие кнопок **Close**, **Resize** или **Stretch** при отсутствии child-форм приводит к исключительной ситуации `NullReferenceException` (поскольку в этом случае свойство `ActiveMDIChild` равно `null`).

Исправление: для кнопок `close2`, `resize2` и `stretch2` положите свойство `Modifiers` равным **`Internal`**, свойство `Enabled` равным **`False`** и измените метод `open1_Click` формы `Form1` (листинг 27.12) и метод `Form2_FormClosed` формы `Form2` (листинг 27.13).

Листинг 27.12. Новый вариант метода `open1_Click` формы `Form1`

```
internal void open1_Click(object sender, EventArgs e)
{
  openFileDialog1.FileName = "";
  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
    Form2 f = new Form2();
    string n = openFileDialog1.FileName;
    Image i = Image.FromFile(n);
    f.pictureBox1.Image = i;
    f.Text = string.Format("{0} ({1}x{2})",
      System.IO.Path.GetFileNameWithoutExtension(n), i.Width, i.Height);
    f.MdiParent = this;
    window1.Visible = true;
    close2.Enabled = resize2.Enabled = stretch2.Enabled = true;
    f.Show();
  }
}
```

Листинг 27.13. Новый вариант метода `Form2_FormClosed` формы `Form2`

```
private void Form2_FormClosed(object sender, FormClosedEventArgs e)
{
  Form1 f = MdiParent as Form1;
  if (f.MdiChildren.Length == 1)
    f.window1.Visible = f.close2.Enabled = f.resize2.Enabled =
      f.stretch2.Enabled = false;
}
```

Результат: если главная форма не содержит ни одной child-формы, то кнопки **Close**, **Resize** и **Stretch** недоступны. Отметим, что скрывать недоступные кнопки быстрого доступа (в отличие от пунктов меню первого уровня) не принято.

Недочет 1: при наведении курсора мыши на любую кнопку быстрого доступа рядом с ним отображается всплывающая подсказка, дублирующая текст на кнопке.

Исправление: в нашем случае нет необходимости в отображении всплывающих подсказок, поскольку сами кнопки содержат текст, поясняющий их назначение. Поэтому отменим возможность отображения всплывающих подсказок для всей панели инструментов, положив свойство `ShowItemToolTips` компонента `toolStrip1` равным **`False`** (см. также комментарий 1).

Недочет 2: для большей наглядности желательно отображать наличие или отсутствие режима масштабирования с помощью выделения кнопки **Stretch**.

Исправление: определите обработчик события Enter для формы Form2 (листинг 27.14).

В метод `stretch1_CheckedChanged` (листинг 27.9) добавьте оператор

```
(MdiParent as Form1).stretch2.Checked = stretch1.Checked;
```

Последний оператор в методе `Form2_FormClosed` (листинг 27.13) измените следующим образом:

```
f.window1.Visible = f.close2.Enabled = f.resize2.Enabled =
  f.stretch2.Enabled = f.stretch2.Checked = false;
```

Примечание

Изменение в методе `Form2_FormClosed` гарантирует, что *недоступная* кнопка **Stretch** никогда не будет изображаться выделенной.

Листинг 27.14. Обработчик `Form2.Enter`

```
private void Form2_Enter(object sender, EventArgs e)
```

```
{
  (MdiParent as Form1).stretch2.Checked = stretch1.Checked;
}
```

Результат: теперь переход кнопки **Stretch** из выделенного в невыделенное состояние и обратно происходит в следующих ситуациях (см. также комментарий 2):

- при щелчке на этой кнопке;
- при выполнении команды меню **Stretch;**
- при переходе к дочерней форме, имеющей другой режим масштабирования.

27.8. Прокрутка изображения с помощью клавиатуры

Определите обработчик события `KeyDown` для формы `Form2` (листинг 27.15).

Листинг 27.15. Обработчик `Form2.KeyDown`

```
private void Form2_KeyDown(object sender, KeyEventArgs e)
{
  if (e.Modifiers != Keys.Control)
    return;
  Point p = AutoScrollPosition;
  Size s = pictureBox1.Size;
  p.X = -p.X;
  p.Y = -p.Y;
  switch (e.KeyCode)
  {
    case Keys.Up:
      p -= new Size(0, 10); break;
    case Keys.Down:
      p += new Size(0, 10); break;
    case Keys.Left:
      p -= new Size(10, 0); break;
    case Keys.Right:
      p += new Size(10, 0); break;
    case Keys.Home:
      p = Point.Empty; break;
    case Keys.PageDown:
      p = new Point(s); break;
    case Keys.End:
      p = new Point(0, s.Height); break;
    case Keys.PageUp:
      p = new Point(s.Width, 0); break;
  }
  AutoScrollPosition = p;
}
```

Результат: прокрутку изображения (если режим масштабирования отключен и child-форма содержит только часть изображения) можно выполнять не только мышью на полосах прокрутки, но и с помощью клавиатуры при нажатой клавише <Ctrl>: клавиши со стрелками обеспечивают прокрутку в указанном направлении, а клавиши <Home>, <End>, <PgUp> и <PgDn> обеспечивают перемещение в один из углов изображения (соответственно, в левый верхний, левый нижний, правый верхний и правый нижний). Необходимость в использовании клавиши <Ctrl> объясняется тем, что обычные клавиши со стрелками уже зарезервированы в MDI-приложении для переключения между дочерними формами (см. разд. 27.1).

Приведем пример работающей программы (рис. 27.6). В окне программы содержатся две дочерние формы с одним и тем же изображением. В левой (активной) форме включен режим масштабирования, правая снабжена полосами прокрутки.

Рис. 27.6. Вид работающего приложения JPEGVIEW