Лекция 9. Технология Windows Forms. Стандартные элементы пользовательского интерфейса

Технология Windows Forms является частью платформы Microsoft .NET Framework и используется для написания программ, которые обычно называют клиентскими Windows-приложениями (альтернативные названия: «прикладное ПО рабочего стола» или просто «стандартные Windows-приложения»). Эти приложения состоят из обычных исполняемых файлов (с расширением `.exe`), иногда в них входят файлы динамических библиотек (с расширением `.dll`) и другие необходимые файлы. Windows Forms считается современной альтернативой прежних методов создания Windows-приложений, таких как использование языка программирования C и «родного» 32-битного интерфейса прикладного программирования для Windows (Win32 API), а также программирование на C++ с использованием классов библиотеки MFC (Microsoft Foundation Classes).

Windows Forms реализована в нескольких DLL-библиотеках, составляющих платформу .NET. Платформа .NET должна присутствовать и на компьютере, где разрабатываются программы Windows Forms, и на машинах, где они выполняются. При необходимости в программах Windows Forms можно использовать Win32 API, но чаще всего библиотек .NET Windows Forms вполне хватает. Большинство этих библиотек — это наборы объектно-ориентированных классов, содержащих обычные поля и методы, а также члены классов — *свойства* (*properties*) и *события* (*events*). Соответственно, язык программирования, в котором используются эти классы, должен поддерживать свойства и события, а также все базовые типы, используемые в этих классах (целые числа, числа с плавающей точкой, строки и др.).

Программы не только задают различные свойства, но ещё должны откликаться на ввод пользователя. Для этого служат *события* (*events*) — это универсальный механизм, позволяющий одному классу (объекту) сигнализировать другому классу (объекту) путём вызова метода. Вызываемый метод называется *обработчиком события* (*event handler*). Обработчик содержит аргументы и возвращаемое значение, которые не переопределяются даже после многократных приведений к типу.

Простое приложение Windows Forms создаётся почти так же легко, как консольное. Проект приложения Windows Forms включает в себя несколько файлов, среди которых можно выделить файл `Program.cs` с кодом, который выполняется первым при запуске приложения, и файлы с кодом форм приложения, в которых заключается практически весь код для работы программы. Для нового проекта файл `Program.cs` выглядит примерно так:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

В этом файле с помощью ключевого слова `using` определяются пространства имён, содержащие часто используемые в этом файле кода классы. Обычно содержимое этого файла изменять не требуется, однако в

большинстве случаев можно оставить пространства имён System и System.Windows.Forms. Ключевое слово namespace определяет имя пространства имён, в котором будут размещены все классы приложения. Для небольших приложений оно обычно является единственным и совпадает с именем проекта. В текущем файле определяется статический класс Program, содержащий единственный метод Main, с которого начинается выполнение программы.

В методе вызываются три метода класса Application, который содержит статические методы и свойства для управления приложением. Метод Run запускает форму, указанную в параметре, и передаёт ей управление. Остальные методы используются для различных целей, удалять их вызовы не рекомендуется. В данном случае создаётся *объект Form* — экземпляр формы, которая описывается в классе Form1. Этот класс создаётся по умолчанию при создании проекта Windows Forms. Его код содержится в двух файлах:

- Form1.cs — содержит код, определяющий программную логику работы формы. Этот файл можно редактировать непосредственно, а также его содержимое частично создаётся дизайнером кода Visual Studio;
- Form1.Designer.cs — содержит код, определяющий внешний вид формы и её компонентов. Этот файл генерируется дизайнером кода Visual Studio, изменять его содержимое вручную обычно не следует.

Имя формы по умолчанию (Form1) рекомендуется переименовать в более осмысленное — особенно если приложение содержит несколько форм. Для этого необходимо в окне обозревателя решений Visual Studio переименовать файл Form1.cs, например, в MainForm.cs, и Visual Studio автоматически переименует связанные файлы, класс и все ссылки на него.

Вызов метода Application.Run с точки зрения Windows-программирования создаёт очередь для получения сообщений, которые Windows отправляет приложениям. Метод Run не возвращает управление программе, пока пользователь не закроет переданный этому методу объект Form. Порядок таков: когда метод Run возвращает управление программе после отображения формы, метод Main завершается, программа прекращает работу и в процессе очистки системы окно уничтожается.

При необходимости перед открытием формы можно программно изменить её свойства. Для этого нужно создать объект Form, затем назначить ему свойства и лишь потом передать объект методу Application.Run:

```
MainForm form = new MainForm();
form.Text = "Моя программа Windows Forms";
form.Width = 600;
Application.Run(form);
```

Text и Width — члены класса Form. Свойство Text ссылается на текст, отображаемый в строке заголовка программы. Width содержит значение ширины окна программы в пикселах. Программа задаёт ширину объекта Form, отличную от заданной по умолчанию, чтобы весь текст заголовка был виден.

Сам класс MainForm является наследником класса Form, который содержим свойства, методы и события, одинаковые для всех форм. Свойства формы можно задавать в конструкторе объекта Form, вызов которого содержится в файле MainForm.cs — и это удобнее, чем редактировать файл Program.cs.

```
using System.Windows.Forms;

namespace WindowsFormsApplication
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();

            Text = "Моя программа Windows Forms";
            Width = 600;
        }
    }
}
```

Ключевое слово `partial` в определении класса позволяет распределить код класса среди нескольких файлов. Так код для создания элементов формы, который генерируется средой Visual Studio, находится в отдельном файле, как было отмечено выше.

Когда Visual Studio отображает внешний вид формы в конструкторе, в правом нижнем углу появляется окно свойств, где можно изменить любые свойства формы, а также, щёлкнув значок молнии, просмотреть события формы. Двойной щелчок по пустому полю рядом с событием создаст пустой метод-обработчик этого события и сгенерирует весь необходимый для этого код. Например, чтобы изменить размер, найдите свойство `Size`, у которого есть два свойства — `Width` (ширина) и `Height` (высота). Размер формы также можно изменять напрямую в окне дизайнера, перемещая один из маленьких квадратиков на границах формы. В любом случае, Visual Studio вставит в файл `MainForm.Designer.cs` код следующего вида:

```
this.ClientSize = new System.Drawing.Size(565, 266);
```

В клиентской области формы приложение отображает визуальную информацию. Программа Windows Forms заполняет клиентскую область текстом и графикой, помещает на неё элементы управления (стандартные и пользовательские) или комбинирует графику с элементами управления. Из элементов управления в программах Windows Forms доступны привычные кнопки, поля ввода, окна списков, полосы прокрутки и более сложные элементы. Клиентские области приложений часто разделяют на логические области, при этом также применяются такие элементы управления, как панели и разделители.

Класс `Control`

Класс `Control` предоставляет большое количество инструментов для создания приложений Windows Forms. `Control` является не только базовым классом для таких элементов управления, как кнопки, деревья, панели инструментов и меню, но и для `Form` — класса, инкапсулирующего главное окно приложения Windows Forms и выполняющего другие функции в диалоговых окнах.

Объект Form по умолчанию содержит строку заголовка с системным меню слева и кнопки свёртки, развёртки и закрытия окна — справа. По периметру окна проходит рамка определения размера окна. Внутри окна расположена клиентская область — в ней отображается информация, и содержатся инструменты для пользователя. Программы могут отображать текст и графическую информацию непосредственно в клиентской области и получать команды с клавиатуры или мыши с применением таких методов, как `OnKeyPress` и `OnMouseDown`. Но в большинстве современных программ в клиентской области располагаются дочерние элементы управления — обработка ввода/вывода информации о действиях пользователя делегируется им. Кроме того, программисты вправе создавать собственные элементы управления, расширяя либо комбинируя имеющиеся.

Практически всё на экране Microsoft Windows состоит из элементов управления. В общем случае, элемент управления можно определить как визуальный объект. Обычно элементы управления занимают прямоугольную область экрана, хотя могут быть непрямоугольными или даже скрытыми и выполнять команды, поступающие с клавиатуры или мыши. Класс `Control` поддерживает множество событий (и соответствующих методов ввода) для обработки ввода информации, например событий клавиатуры `KeyDown`, `KeyUp` и `KeyPress` и мыши — `MouseDown`, `MouseUp` и `MouseMove`. Также элементы управления должны «перерисовывать» себя на экране — для этого служит метод `OnPaint`.

Превращая информацию об операциях пользователя в простые события, элементы управления служат уровнем абстрагирования между пользователем и приложением. Например, для нормальной работы элемента `Button` достаточно определить отображаемый на кнопке текст и обработчик события `Click`. Об остальном позаботится элемент управления.

Одно из самых важных определяемых классом `Control` свойств — `Parent`; оно указывает на другой объект типа `Control`. В процессе исполнения приложения на экране отображаются только элементы управления с корректно определённым свойством `Parent`. Положение элемента всегда задаётся относительно «родителя» и отображается на поверхности последнего. Часть элемента управления, выступающая за края родителя, не видна.

Класс `Form` является исключением из этого правила. Значение свойства `Parent` объекта типа `Form` обычно равно `null`, то есть родительским свойством формы является рабочий стол. Однако объект Form, у которого

значение свойства TopLevel равно false (то есть родитель формы — не рабочий стол), может быть потомком любого другого элемента управления. Такая методика используется в приложениях с *многодокументным интерфейсом* (*Multiple Document Interface — MDI*). В классе Form есть свойство Owner, в котором можно задать другой объект типа Form. Форма на экране всегда отображается поверх своего владельца, и если последний сворачивается или закрывается, то же происходит с формой. Этот механизм полезен в немодальных диалоговых окнах. Владелец немодального диалогового окна должен сопоставляться создавшему его приложению.

В простейшем случае, объект типа Form создает набор элементов управления и «прописывает» себя в свойстве Parent элементов управления. Вот соответствующий код в конструкторе форм:

```
Button button = new Button();
button.Parent = this;
```

Оператор, присваивающий значение свойству Parent кнопки, функционально эквивалентен оператору:

```
this.Controls.Add(button);
```

Controls — это свойство, определяемое классом Control и унаследованное классом Form. Это объект типа Control.ControlCollection — то есть, экземпляр класса ControlCollection, определенного в классе Control. Свойство Controls содержит всех потомков класса Control. Метод Add добавляет элемент управления в набор, а Remove — удаляет:

```
this.Controls.Remove(button);
```

Этот оператор функционально эквивалентен коду:

```
button.Parent = null;
```

Свойство Controls может представляться в виде массива, где нумерация элементов начинается с нуля. По номеру можно обратиться к любому элементу набора и получить ссылку на объект типа Control. Если форма «знает», что, к примеру, это объект Button, его можно привести к объекту Button:

```
Controls[3] as Button
```

или

```
(Button)Controls[3]
```

Между этими двумя вариантами приведения есть разница. Если Controls[3] не является объектом Button, выражение as возвратит null. Во втором случае программа вернёт исключение. В программе можно проверить, является ли конкретный Control объектом Button, воспользовавшись выражением is:

```
if (Controls[3] is Button)
{
    Button btn = Controls[3] as Button;
}
```

Форма позволяет перечислить все элементы управления своего набора, используя цикл for:

```
for (int i = 0; i < Controls.Count; i++)
{
    Control control = Controls[i];
}
```

Для этого также часто задействуют цикл foreach:

```
foreach (Control control in Controls)
{
}
```

Хотя любой элемент управления может быть родителем по отношению к другим элементам, в действительности, немногие наследники Control используются в качестве родителей. Обычно это Form, Panel и производные классы, а также GroupBox. Вообще говоря, элементы управления — это дочерние элементы формы, но иерархия «родитель—потомок» может распространяться гораздо дальше. Например, Form может содержать в своей клиентской области несколько элементов управления Panel, а эти панели могут иметь собственные дочерние элементы управления.

Элементы управления, производные от Panel и ContainerControl (включая Form), самостоятельно управляют фокусом среди своих потомков. Как правило, перемещение между элементам управления выполняется клавишей Tab. Фокус ввода способны получать только элементы управления с заданным свойством TabStop. Порядок получения фокуса задается свойством Tablndex производных элементов управления, или выполняется в так называемом z-порядке, то есть в порядке, в котором элементы были добавлены в набор.

Дочерние элементы управления одного родителя (и, поэтому, члены одного набора) называются *родственными*.

По умолчанию все элементы управления видимы и активны. Если свойству Visible присвоить значение false, элемент управления исчезнет с поверхности своего родителя. Он никуда не пропадёт и по-прежнему будет оставаться членом родительского набора элементов управления, но картина будет такой, как будто его нет. Все потомки невидимого элемента управления также невидимы.

Однако наиболее часто меняют свойство Enabled. Элемент со свойством Enabled равным false отображается на экране, но выглядит обесцвеченным, или отключенным. Такой элемент управления не получает фокус ввода и не реагирует на команды клавиатуры или мыши.

Часто для активизации и отключения элементов управления используются параметры других элементов. Например, в диалоговом окне открытия файла кнопка Открыть будет отключена, пока пользователь не задаст имя файла.

Каждому элементу управления задаётся определённый размер и положение относительно левого верхнего угла его родителя. Расположение определяется свойством Location — объектом Point с двумя свойствами — X и Y. Координаты задаются в пикселах, и определяют положение верхнего левого угла элемента управления относительно верхнего левого угла родителя. Размер элемента задаётся свойством Size — объектом типа Size со свойствами Width и Height.

Вот код, задающий кнопку с положением в точке (50,100) и размерами 75 пикселов по ширине и 25 — по высоте:

```
button.Location = new Point(50, 100);
button.Size = new Size(75, 25);
```

Конечно, существует сложности, связанные с работой программы на различных системах с различными разрешениями экрана. Кроме того, может потребоваться перевод программы на другой язык. В одних языках слова короче, в других — длиннее, и это влияет на размер элементов управления, отображающих текст. Усложнение элементов управления приводит к тому, что задавать должный размер становится всё труднее. Проще позволить элементам управления самостоятельно определять нужный размер и располагаться в форме динамически в зависимости от требуемого размера.

У Control есть свойство «только для чтения» PreferredSize, задающее размер элемента управления в зависимости от его содержимого и используемого шрифта. Очень мощное изменяемое свойство — AutoSize. Обычно в большинстве элементов управления его значение равно false, но если ему присвоить значение true, свойство Size примет значение PreferredSize.

У некоторых элементов управления есть свойство AutoSizeMode, которому присваивается одно из значений перечисления AutoSizeMode. Параметры GrowOnly и GrowAndShrink определяют, как элемент управления ведёт себя во время исполнения, когда его содержимое меняется. Свойства Padding и Margin используются для динамического размещения.

Ещё два свойства, важных для процесса разметки формы — Dock и Anchor. Первое используется для растягивания и выравнивания элемента управления по определённому краю родителя. Для этого свойству задают одно из значений перечисления DockStyle — Left, Right, Top, Bottom или Fill. Например, совершенно естественно, что у панели инструментов значение свойства Dock равно DockStyle.Top (то есть панель расположена вверху родительской формы), а у строки состояния — DockStyle.Bottom. DockStyleFill заставляет элемент управления заполнить всю клиентскую область родителя.

Свойство Anchor заставляет элемент управления располагаться на определённом постоянном расстоянии от одной из сторон родителя при изменении размеров последнего. Свойству может присваиваться любая комбинация членов перечисления AnchorStyles. По умолчанию используется AnchorStylesLeft | AnchorStylesRight, то есть при изменении размера родителя элемент управления остаётся в том же положении относительно его левой и правой сторон.

У всех элементов управления есть свойство Text, хотя некоторые из них (например, полоса прокрутки) не отображают никакого текста. Для объекта Form, свойство Text — это текст, отображаемый в строке заголовка формы.

Элемент управления наследует исходные свойства Font, BackColor и ForeColor у своего родителя. При изменении свойств родителя свойства наследников также изменяются. Однако если элементам управления явно задать другие свойства, они сохранят их и перестанут зависеть от родителей.

В качестве значения свойства Font задают объект типа Font. Например, следующее выражение задаёт шрифт кнопки: курсив Times New Roman размером 24 пункта.

```
button.Font = new Font("Times New Roman", 24, FontStyle.Italic);
```

Свойства BackColor и ForeColor задают, соответственно, цвет фона и изображения элемента управления. Цветом изображения часто является цвет текста элемента управления. Некоторые элементы управления игнорируют такие свойства либо используют их другим образом. В качестве значения этим свойствам можно присвоить любой созданный объект Color или один из статических членов структуры Color:

```
button.ForeColor = Color.HotPink;
```

По умолчанию свойствам BackColor и ForeColor элементов управления задаются значения SystemColors.Control и SystemColors.ControlText, соответственно. Объекту Form также по умолчанию назначаются эти цвета. Поскольку SystemColors.Control — это обычно серый цвет, фон клиентской области формы — серый, что обычно для диалоговых окон всех основных приложений. Если нужно, чтобы форма отображала чёрный текст на белом фоне, введите в конструктор форм код:

```
BackColor = SystemColors.Window;
ForeColor = SystemColors.WindowText;
```

Чтобы не отменять персональные параметры пользователей, предпочитающих экран с белым текстом на черном фоне, лучше использовать системные, а не черно-белые цвета. В этом случае у таких пользователей SystemColors.Window будет черным, а SystemColors.WindowText — белым.

Свойство Backgroundlmage позволяет задать изображение (либо растр или метафайл) на фоне элемента управления. Свойство BackgroundlmageLayout позволять справляться с ситуациями, когда изображение не совпадает с элементом управления по размеру.

У формы может быть много дочерних элементов управления. В качестве примера возьмём диалоговое окно с набором элементов управления и кнопкой OK. Вполне может быть так, что диалоговое окно игнорирует любые действия пользователя, пока не нажата кнопка OK. В этом случае, диалоговое окно опросит все элементы управления на предмет наличия текста, выбранных флажков и т. п. Программе, созданной таким образом, нет необходимости устанавливать обработчики событий для всех остальных элементов управления, а нужен способ доступа ко всем элементам управления.

Один из способов заключается в хранении всех объектов управления как полей, в этом случае элементы управления доступны по соответствующему классу. Однако есть и другие способы. Из класса форм, через набор

Controls доступны все дочерние элементы управления. Выделить конкретный элемент из набора можно по его типу, свойствам Text или Name, которые специально созданы для этих целей. Например, можно задать элементу уникальное имя и потом обратиться к этому элементу по заданному имени из другого метода в этой форме:

```
button.Name = "ok";

...

Button button = Controls["ok"] as Button;
```

Набор Controls возвращает объект типа Control, а выражение as приводит его к объекту типа Button. Свойство Controls — это экземпляр класса ControlControlCollection, а этот класс предлагает множество методов, таких как Contains и Find, позволяющих управлять набором дочерних элементов управления.

Произвольные объекты можно подключить к элементу управления при помощи свойства Tag. Оно полезно при обозначении набора сходных элементов управления. Например, при использовании элементов управления с независимой фиксацией для курсивного и полужирного начертания шрифта, можно обозначить такой элемент управления, присвоив свойству Tag значение, равное FontStyleItalic и FontStyleBold.

Статические элементы управления

Статические элементы Microsoft Win32 API позволяют выводить текстовую и другую информацию на форму. Содержимое этих элементов обычно задаётся при проектировании формы, но также его можно изменять программно. Пользователь приложения не может изменять их содержимое, однако, эти элементы могут обрабатывать события мыши.

Группа элементов управления (GroupBox)

Группа элементов управления по периметру выделена линией и вверху содержит текст, определённый в свойстве Text. Рамка группы темнее цвета фона.

Обычно группы используются в качестве «родителя» *кнопок-переключателей* (*radio buttons*). Все переключатели внутри одной группы взаимоисключающие. Нажатие клавиш со стрелками перемещает фокус ввода и метку выбора между переключателями. GroupBox не является потомком класса ContainerControl, поскольку в последнем навигация выполняется клавишей Tab.

Переключатели не обязательно являются потомками GroupBox. Общим родителем группы взаимоисключающих переключателей может служить любой элемент управления. Логикой активизации переключателей управляет не группа, а сами переключатели.

Метка (Label)

Label — это элемент управления, отображающий нередактируемый текст. Сам текст задаётся в свойстве Text. Хотя метка может отображать текст в нескольких строках, она не выводит полос прокрутки, если текст не умещается в элементе управления. Если в элементе управления надо разместить объёмный текст, но делать его редактируемым нельзя, используйте элемент управления TextBox со свойством Readonly равным true.

Помимо текста, метка может отображать объект Image, который можете быть как объектом System.Drawing.Bitmap, так и System.Drawing.Imaging.Metafile. Один из способов заключается в присвоении свойству Image загруженного файла:

```
label.Image = Image.FromFile("SillyCat.jpg");
```

Кроме этого, можно использовать изображения, заданные в качестве ресурсов и прикреплённые к исполняемому файлу программы.

Метка-ссылка (LinkLabel)

LinkLabel по функциональности он ближе к Button, но наследуется от Label. Этот элемент управления отображает текстовую строку, сильно или немного выделенную, чтобы показать, что щелчок этого элемента вызывает какое-то действие, например запуск приложения или открытие веб-страницы. Вдобавок к назначению стандартных значений свойств BackColor и ForeColor, в LinkLabel определены четыре свойства цвета:

- LinkColor — цвет обычной ссылки, по умолчанию синий;
- DisabledLinkColor — цвет неактивной ссылки, по умолчанию серый;
- ActiveLinkColor — цвет при активизации ссылки, по умолчанию красный;
- VisitedLinkColor — цвет выбранной ранее, по умолчанию пурпурный.

Свойство LinkArea элемента LinkLabel — это объект типа LinkArea, представляющий собой структуру из двух целых свойств — Start и Length. Допустим, свойство Text элемента LinkLabel определено так:

```
link.Text = "Нажмите здесь, чтобы открыть страницу";
```

Пусть слово «здесь» надо сделать ссылкой. Для этого нужно задать значение свойства LinkArea так, чтобы ссылка начиналась с восьмого символа (начиная с нуля) и была длиной пять символов:

```
link.LinkArea = new LinkArea(8, 5);
```

Вид этой части символьной строки определяется свойством LinkBehavior, которому присваивается одно из значений перечисления LinkBehavior:

- AlwaysUnderline — текст ссылки подчёркивается всегда;
- HoverUnderline — текст ссылки подчёркивается, только когда мышь находится на тексте ссылки;
- NeverUnderline — текст ссылки никогда не отображается подчёркнутым;
- SystemDefault — поведение данной настройки зависит от параметров, заданных с помощью диалогового окна «Свойства обозревателя» панели управления или Internet Explorer.

По умолчанию назначается SystemDefault, повторяющее поведение ссылки в соответствии с пользовательской настройкой Microsoft Internet Explorer.

Одной метке можно назначить несколько ссылок. В этом случае применяется свойство LinkLabel по имени Links. Это объект LinkLabelLinkCollection, представляющий собой набор объектов LinkLabelLinks. В качестве ссылки может как путь к исполняемому файлу, так и URL веб-страницы. В последнем случае запускается браузер, и происходит переход по указанному в ссылке адресу.

Графическое окно (PictureBox)

Элемент управления PictureBox служит для отображения изображения, представляющего собой растр или метафайл. Изображение задается свойством Image. Можно создать набор ImageList различных изображений и присвоить его свойству ImageList объекта PictureBox. Изображение задаётся либо по ImageIndex (числовому указателю изображения в списке), либо по ImageKey (ссылке на изображение в виде текстовой строки).

Свойство SizeMode элемента PictureBox регулирует «растягивание» изображения в рамках родительского элемента управления. Свойству SizeMode задают значение одного из указанных ниже членов перечисления PictureBoxSizeMode. Первые три члена не меняют заданного в пикселах размера изображения — оно отображается без растяжения:

- Normal — изображение размещается в верхнем левом углу графического окна;
- CenterImage — изображение размещается по центру элемента управления;
- AutoSize — графическое окно принимает размер изображения;
- StretchImage — изображение принимает размер элемента управления;
- Zoom — изображение масштабируется без искажений.

У элемента PictureBox есть несколько свойств, позволяющих задать URL-адрес или имя локального файла при помощи методов Load или LoadAsync. Метод LoadAsync загружает файл в дополнительном потоке и информирует о завершении событием LoadCompleted. Можно также задать InitialImage — «картинку», отображаемую во время загрузки нужного изображения, и ErrorImage — изображение, выводимое в случае ошибки загрузки. Событие LoadProgressChanged предоставляет информацию о загрузке изображения в процентах. Его можно использовать для отображения индикатора выполнения — элемента управления ProgressBar.

Индикатор хода процесса (ProgressBar)

Элемент ProgressBar выводит полосу прокрутки, которую удобно использовать для отслеживания процесса выполнения какой-либо продолжительной операции.

Основные свойства ProgressBar — это целочисленные Minimum, Maximum и Value. По мере выполнения процесса свойству Value присваивают возрастающие значения в диапазоне от Minimum до Maximum.

Есть другой способ: увеличивать значение Value, передавая целочисленный аргумент методу Increment. Также можно вызывать метод PerformStep, увеличивающий свойство Value на величину, заданную в свойстве Step.

Свойство Style принимает одно из значений перечисления ProgressBarStyle: Blocks (по умолчанию), Continuous или Marquee. При выборе Marquee отображается просто анимация, ход процесса.

Кнопки и переключатели

Элемент управления «кнопка» называется так не потому, что его можно щёлкать, — в конечном итоге, все элементы по щелчку инициируют событие Click, а главным образом по той причине, что при щелчке он визуально ведёт себя как кнопка.

Все три типа кнопок происходят от абстрактного класса ButtonBase. Обычно Button используется для инициирования действий, CheckBox — для установки и снятия флажков, а RadioButton обычно служит для выбора одного из нескольких взаимоисключающих вариантов.

На кнопке как правило присутствует текст, но некоторые из них — особенно экземпляры класса Button — содержат изображения, с текстом или без. Класс ButtonBase содержит свойство Image, наследуемое кнопками всех трёх типов и позволяющее назначить отображаемый на кнопке растр или метафайл. Кроме этого, можно задать свойство ImageList объекта ButtonBase и задать конкретное изображение, используя свойства ImageIndex или ImageKey.

Если кнопка должна отображать и текст, и изображение, можно задать определённое в объекте ButtonBase свойство TextImageRelation, которому присваивают одно из значений перечисления TextImageRelation, состоящего из членов ImageAboveText, ImageBeforeText, TextAboveImage, TextBeforeImage и Overlay.

Свойства TextAlign и ImageAlign задают положение текста или изображения в элементе управления. Оба принимают значения перечисления ContentAlignment, состоящего из девяти членов — комбинации Top, Middle и Bottom с Left, Center и Right. Значение по умолчанию обоих свойств — ContentAlignment.MiddleCenter.

Кнопка (Button)

По сравнению с ButtonBase класс Button почти ничего не добавляет нового. В нём практически всегда определяется обработчик события Click. Единственное исключение — присвоение свойству DialogResult значения из перечисления DialogResult (обычно DialogResult.OK или DialogResult.Cancel).

Флажок (CheckBox)

Обычно элемент управления CheckBox — это небольшой прямоугольник, расположенный слева от текста. Однако если свойству Appearance задать значение Appearance.Button (по умолчанию Appearance.Normal), элемент управления будет схож с кнопкой, за исключением того, что будет по щелчку «залипать» и «отлипать». Вид элемента управления CheckBox можно изменить, задав свойству CheckAlign значение из перечисления ContentAlignment. По умолчанию используется значение ContentAlignment.MiddleLeft, то есть флажок по вертикали выравнивается по оси текста и располагается слева от него.

Булево свойство Checked указывает текущее состояние флажка — «установлен» или «сброшен». Событие CheckedChanged информирует об изменении состояния.

По умолчанию значение AutoCheck равно true, то есть по команде с клавиатуры или по щелчку мыши кнопка автоматически изменяет свое состояние и инициирует событие CheckedChanged. Если отклик необходим при любом изменении состояния флажка (например, для активизации и отключения других элементов управления), установите обработчик события CheckedChanged. Если отклик для каждого изменения не нужен, обычно достаточно получить значение свойства Checked, когда пользователь щелчком по кнопке ОК закрывает форму.

Свойству AutoCheck обычно задают false, когда планируют захватывать событие Click и управлять установкой/сбросом флажка самостоятельно, задавая значение свойства Checked из программы. Такое изменение свойства Checked также инициирует событие CheckedChanged.

Иногда недостаточно одного состояния флажка — установленый или сброшенный, — требуется промежуточное состояние. Допустим, CheckBox управляет курсивным начертанием текста. Если в выбранном тексте есть и курсив, и обычный шрифт, CheckBox должен находиться в «неопределённом» состоянии. (С другой стороны, если выбранный текст не может выделяться курсивом, CheckBox должен быть отключён.)

Для использования этого третьего состояния, прежде всего, надо булеву свойству ThreeState присвоить значение true. Вместо свойства Checked нужно задействовать CheckState. Это значение из перечисления CheckState, состоящего из членов Unchecked, Checked и Indeterminate. Также, вместо установки обработчика CheckedChange, используют CheckStateChanged. Если вызвать метод EnableVisualStyles класса Application промежуточное состояние будет выглядеть как небольшой квадрат, в противном случае в элементе управления будет отображаться серый флажок.

Если AutoCheck равно true, по щелчку элемент управления будет по циклу менять свое состояние: «установлен-сброшен-неопределённый». Если необходим другой порядок, присвойте AutoCheck значение false и вручную закодируйте в обработчике события Click порядок смены значений CheckState.

Переключатель (RadioButton)

Этот элемент управления обычно представляет собой кружок, расположенный слева от текста. Как и у Checkbox, у класса RadioButton есть свойство Appearance, которому можно задать значения Appearance.Button — тогда элемент управления выглядит как кнопка, а также свойство CheckAlign, позволяющее менять положение кружка относительно текста.

Как и у CheckBox, у RadioButton есть булево свойство Checked, при изменении которого инициируется событие CheckedChanged.

По умолчанию значение AutoCheck равно true, и в этом отличие RadioButton от CheckBox. Если значение AutoCheck равно true и RadioButton «включен», щелчок кнопки ничего не изменит. Если AutoCheck равно true и RadioButton «выключен», щелчок мышью его «включит». Вдобавок, все «родственные» переключатели «выключатся». Таким образом, в каждый момент включен всего лишь один из группы переключателей RadioButton. Обычно наборы взаимоисключающих кнопок-переключателей создаются как потомки GroupBox, хотя «родитель» у них может быть любой.

Обратите внимание, что «родственные» переключатели поддерживают ввод с клавиатуры: переключение осуществляется клавишами со стрелками.

Как и в ситуации с CheckBox, если не нужна немедленная реакция на изменения, нет необходимости устанавливать обработчик события CheckedChanged. Однако если потребуется определить, какой из переключателей выбран, надо в цикле проверить свойства Checked всех родственных переключателей.

Если AutoCheck равно false, можно обрабатывать событие Click и самостоятельно включать/отключать переключатели. При желании можно сделать включенными несколько переключателей, но пользователя такая ситуация наверняка поставит в тупик.

Полоса прокрутки

Полосы прокрутки обычно устанавливаются с правой стороны или снизу таких элементов управления как ListBox, ComboBox и TextBox (при использовании в многострочном режиме, как в Блокноте Windows). Эти элементы управления автоматически отображают полосы прокрутки. Производные от ScrollableControl элементы управления, в том числе Form и Panel, также отображают полосы прокрутки, когда их свойство AutoScroll приравнено true, а дочерние элементы управления организованы так, что занятая ими область выходит за рамки родителя.

Полосы прокрутки можно создавать не только автоматически. Теоретически, полоса прокрутки позволяет выбирать определенное целое число из непрерывного ряда целых чисел. Указатель полосы прокрутки иллюстрирует положение в этом диапазоне. Поведение элемента управления «ползунок» (TrackBar) похоже, но отлично по виду.

У TrackBar есть свойство Orientation, определяющее положение элемента управления — горизонтальное или вертикальное. В ответ на действия пользователя оба элемента управления меняют позицию указателя автоматически, без вмешательства программными методами.

Горизонтальная и вертикальная полосы прокрутки (ScrollBar)

Наиболее важные свойства ScrollBar — Minimum, Maximum, SmallChange, LargeChange и Value. Все они — целые числа. Программа задает свойствам начальные значения, а свойство Value можно изменить щелчком или управляя полосой прокрутки.

Щелчки стрелок по сторонам полосы прокрутки увеличивают или уменьшают Value с шагом, определённым в SmallChange (это свойство почти всегда равно 1); щелчок в области между стрелкой и указателем изменяет Value с большим шагом, который задан в свойстве LargeChange. Указатель также можно перемещать мышью. Если полоса прокрутки в фокусе ввода, она реагирует на клавиши со стрелками, а также клавиши Page Up, Page Down, Home и End.

Диапазон значений Value не лежит в пределах от Minimum до Maximum — Value меняется от Minimum до (Maximum – LargeChange + 1). Чтобы понять, почему так происходит, посмотрим на работу текстового редактора. Допустим, в документе 1000 строк, пронумерованных от 0 до 999, а окно документа вмещает 20 строк. Используем эти значения для определения свойств полосы прокрутки: Mininum сделаем равным 0, Maximum — 999 и LargeChange — 20. Используем свойство Value полосы прокрутки для определения строки документа, отображаемой вверху окна. При прокрутке документа вниз, Value будет равняться (Maximum – LargeChange + 1) или 980. Строка экрана 980 будет расположена вверху, а строка 999 — внизу окна.

Преимущество этой схемы в том, что размер указателя можно изменять пропорционально LargeChange и Maximum, таким образом наглядно показывая, какая часть всего документа отображается на экране.

ScrollBar реализует два важных события — ValueChanged и Scroll. ValueChanged обрабатывать просто; оно инициируется (как ясно из самого названия) при изменении свойства Value. Получить новое значение в обработчике можно так:

```
void ScrollOnValueChanged(object objSrc, EventArgs args)
{
    int iValue = ((ScrollBar)objSrc).Value;
}
```

Однако бывает, что ValueChanged не предоставляет достаточно информации. Если «захватить» указатель полосы прокрутки мышью и быстро перемещать туда-обратно, программе не будет хватать времени для обработки данных. В таком случае можно задействовать событие Scroll. Обработчики Scroll должны определяться в соответствии с делегатом ScrollEventHandler. Обработчику события предоставляется объект типа ScrollEventArgs, имеющий четыре свойства — OldValue, NewValue, ScrollOrientation и Type.

В сущности, в программе, где обрабатывается событие Scroll, можно запретить реакцию указателя полосы прокрутки на действия пользователя. Для этого свойству NewValue можно задать значение OldValue (которое также является текущим значением свойства Value). Свойство ScrollOrientation (ориентация полосы прокрутки) — это член перечисления ScrollOrientation, имеющего два значения — HorizontalScroll (горизонтальная) и VerticalScroll (вертикальная). Это свойство позволяет использовать один обработчик событий для горизонтальной и вертикальной прокрутки, при этом легко различая их ориентацию.

Свойство Type — это объект типа ScrollEventType, определяющего порядок взаимодействия пользователя с полосой прокрутки. У объекта ScrollEventType девять членов:

- SmallDecrement — мышь: верхняя или левая кнопка со стрелкой; клавиатура: клавиша со стрелкой «вверх» или «влево»;
- SmallIncrement — мышь: нижняя или правая кнопка со стрелкой; клавиатура: клавиша со стрелкой «вправо» или «вниз»;
- LargeDecrement — мышь: левая или верхняя область; клавиатура: клавиша Page Up;
- LargeIncrement — мышь: правая или нижняя область; клавиатура: клавиша Page Down;
- ThumbPosition — кнопка мыши, не удерживающая указатель полосы прокрутки;

- ThumbTrack — нажатое состояние кнопки мыши на указателе полосы прокрутки или перемещение указателя;
- First — клавиатура: клавиша Home;
- Last — клавиатура: клавиша End;
- EndScroll — прокрутка завершена.

Программа с медленным откликом полосы прокрутки может игнорировать все за исключением значения ScrollEventType.EndScroll свойства Type.

Например, если щёлкнуть правую (нижнюю) стрелку полосы прокрутки и задержать её в нажатом состоянии, полоса прокрутки будет инициировать попеременно два события для каждого движения указателя: событие Scroll со свойством Type равным ScrollEventType.SmallIncrement и NewValue, со значением, большим свойства Value. Если NewValue не изменяется программой (стандартный случай), инициируется событие ValueChanged. Когда указатель достигает MaxValue, события ValueChanged больше не инициируются, а инициирование события Scroll продолжится. В конце концов, когда пользователь отпустит кнопку мыши, будет инициировано последнее событие Scroll со свойством Type равным ScrollEventType.EndScroll и NewValue равным Value.

Если захватить и переместить указатель мышью, полоса прокрутки будет инициировать события Scroll и ValueChanged. Об освобождении указателя сигнализируют два последних события Scroll со свойствами Type, равными ScrollEventType.ThumbPosition и ScrollEventType.EndScroll.

Ползунок (TrackBar)

Ползунок аналогичен двум другим элементами управления прокруткой, но выглядит и действует чуть иначе. Как и Scrollbar, у класса TrackBar есть свойства, выраженные целыми числами — Minimum, Maximum, SmallChange, LargeChange и Value. Minimum и Maximum определяют диапазон значений Value.

Класс TrackBar реализует и вертикальный, и горизонтальный ползунки — ориентация задается свойством Orientation. Соответствующее перечисление Orientation содержит два члена — Horizontal и Vertical.

Ползунок представляет собой шкалу с делениями. Свойство TickStyle определяет стиль делений, который выбирается из одноименного перечисления: None, TopLeft, BottomRight и Both. Значение по умолчанию — TickStyle.BottomRight, то есть метки делений отображаются под горизонтальным или справа от вертикального ползунка.

Целочисленное свойство TickFrequency определяет цену деления. Общее количество делений определяется по формуле: (Maximum – Minimum + 1) / TickFrequency.

По умолчанию цена деления равна 1, поэтому нужно быть аккуратным: если, например, увеличить значение Maximum по умолчанию с 10 на 100, метки, скорее всего, сольются в одну чёрную полосу.

Как и полоса прокрутки, ползунок реализует события Scroll и ValueChanged. Однако при обработке ввода между этими событиями различия не делают — они следуют один за другим и предоставляются в простых объектах EventArgs. При изменении свойства Value из программы, инициируется событие ValueChanged, но не Scroll.

Ползунки лучше всего подходят для выбора значения из дискретного диапазона, поэтому неплохо размещать метку, указывающую выбранное значение. Есть и альтернативное решение для выбора значения из дискретного набора — это элемент управления «наборный счетчик» (*spin control*).

Элементы управления с поддержкой редактирования текста

В Windows Forms есть несколько элементов управления, служащих для ввода или редактирования текста.

Самое важное свойство элемента с поддержкой редактирования текста — это Text, содержащее текст, отображаемый в элементе управления. Программа инициализирует текст в элементе управления, простым определением значения Text, и получает текст, введённый или измененный пользователем, обращением к свойству Text. Если txtbox — объект типа TextBox, удалить из него текст просто:

```
txtbox.Text = "";
```

А добавляют строку в конец существующего текста так:

```
txtbox.Text += " and that's the truth";
```

Многие важные свойства этих элементов управления определены в абстрактном классе TextBoxBase. Свойство Multiline определяет, может ли элемент управления получать и отображать многострочный текст. По умолчанию оно равно false в TextBox и MaskedTextBox, и true — в RichTextBox. Свойство WordWrap (перенос по словам) применимо только к многострочным элементам управления и по умолчанию равно true. Чтобы элемент управления отображал нередактируемый текст, свойству Readonly присваивают значение true.

Выбор текста выполняется с клавиатуры или мыши. Свойства SelectedStart и SelectedLength — целые числа, указывающие позицию первого символа выбранного текста и длину выбранного отрезка в символах соответственно. SelectedText — это и есть выбранный текст. TextBoxBase реализует собственные операции вырезки, копирования, вставки и отмены на основе стандартных комбинаций клавишей — Ctrl + X, Ctrl + C, Ctrl + V и Ctrl + Z соответственно. Кроме этого, методы Cut, Copy, Paste и Undo выполняют эти операции с буфером обмена программно, обычно в ответ на выбор команды меню.

Самое важное событие в этих элементах управления — TextChanged; оно инициируется при изменении свойства Text.

Текстовое окно с маской (MaskedTextBox)

Элемент MaskedTextBox создан для ввода одной строки текста в заранее заданном формате, например телефонных номеров, денежных сумм, дат или адресов электронной почты. Понять, как использовать этот элемент управления, проще всего из описания свойства Mask, представляющего собой символьную строку, определяющую формат вводимой в поле строки.

При попытке ввести не предусмотренный маской текст инициируется событие MaskInputRejected, вместе с которым передается объект MaskInputRejectedEventArgs, имеющий два свойства: Position — позиция символа, не совпадающего с маской, и RejectionHint — текстовая строка, отображаемая (например, в элементе управления Label) для прочтения пользователем.

Текстовое поле (TextBox)

Это простейший элемент управления с поддержкой редактирования текста, но вместе с тем он достаточно сложен, чтобы стать основой Блокнота Windows.

TextBox добавляет всего несколько свойств к имеющимся в TextBoxBase. Наверное, самое важное из них — это ScrollBars, которому присваиваются значения из перечисления ScrollBars.Vertical, ScrollBars.Horizontal или ScrollBars.Both. Значение по умолчанию — ScrollBars.None, то есть, полосы прокрутки не отображаются, даже если этого требует длина текста. (Если значение свойства WordWrap равно true, горизонтальные полосы прокрутки не отображаются независимо от значения ScrollBars).

Если TextBox используется для ввода пароля, нужно указать в PasswordChar символ, который будет отображаться при вводе. В этом отношении также полезно свойство CharacterCasing. Если присвоить этому свойству значение из перечисления CharacterCasing.Lower или CharacterCasing.Upper, вводимые символы будут приводиться в нижний или верхний регистр соответственно.

Поле ввода с форматированием (RichTextBox)

В элементе RichTextBox текст хранится в поддерживающем стилевое оформление формате RTF (Microsoft Rich Text Format). Этот элемент управления является основой программы Windows WordPad.

RichTextBox по умолчанию многострочный. Как и в TextBox, в RichTextBox есть свойство ScrollBars, но здесь ему присваивается одно из значений перечисления RichTextScrollBars. По умолчанию это RichTextScrollBars.Both, предписывающий горизонтальные и вертикальные полосы прокрутки, но только если они нужны. Остальные члены перечисления — None, Horizontal, Vertical, ForcedHorizontal, ForcedVertical и ForcedBoth. Значения с префиксом Forced требуют отображать полосы прокрутки, даже если они не обязательны.

RichTextBox разрешает определять программно форматирование только выделенного в данный момент текста. Такое форматирование выполняется путём определения свойств, начинающихся со слова Selection. В некоторых случаях эта схема очень удобна, например когда пользователь выделяет текст, а затем выбирает

Font из меню Format. В программе создается новый объект Font, например по имени fnt, и присваивается свойству SelectionFont элемента управления RichTextBox:

```
rtb.SelectionFont = fnt;
```

Однако если потребуется определить цветовое оформление не выделенного текста (например для визуального выделения ошибки в текстовом редакторе, используемом как простейший редактор кода), программисту придется изрядно попотеть. Нужно сохранить значения определяющих выделение свойств SelectionStart и SelectionLength. Затем нужно методом Select задать новое выделение — текст, цвет которого следует изменить. Выделение окрашивается присвоением свойству SelectionColor соответствующего объекта Color. Процедура завершается методом Select, восстанавливающим исходное выделение.

SelectionBackColor и SelectionColor — это свойства объекта Color, определяющие цвет фона и текста выделенного участка соответственно. SelectionFont — это объект типа Font. Ещё один элемент форматирования символов — SelectionCharOffset, смещение уровня текста, выраженное в пикселах. Положительное значение используется для верхних, а отрицательное — для нижних индексов.

Все остальное относится к форматированию абзацев. SelectionAlignment принимает значения из перечисления HorizontalAlignment, состоящего из членов Left, Right и Center. Выравнивания по правому и левому краям одновременно не предусмотрено. SelectionBullet — булево свойство, задающее маркер перед каждым абзацем. SelectionIndent, SelectionRightIndent и SelectionHangingIndent — отступ в пикселах. SelectionRightIndent отмеряется от правой стороны элемента управления. Отступ первой строки абзаца указывается в SelectionIndent и отмеряется от левой стороны элемента управления. SelectionHangingIndent — «висячий отступ», то есть отступ остатка абзаца по отношению к SelectionIndent. (Здесь есть некоторое отличие от определения отступа в стандартных текстовых редакторах). SelectionTabs — это массив чисел, определяющих позиции табуляции.

В свойстве Text объекта RichTextBox задается только «пустой» текст без каких-либо RTF-тэгов. Для работы с полнофункциональным RTF-текстом служит свойство Rtf. В RichTextBox предусмотрены встроенные методы ввода/вывода файлов — LoadFile и SaveFile.

Список и поле со списком

В наиболее общем и абстрактном виде элемент управления ListBox позволяет пользователю выбрать элемент из списка из одного или нескольких элементов. В дополнение к этому ComboBox позволяет вводить текст, не содержащийся в списке. Эти элементы управления наследуются от абстрактного класса ListControl.

В последнее время элемент управления ListBox уступил место ComboBox, занимающему меньше экранного пространства. Единственная функция, которая есть у ListBox, но нет у ComboBox — возможность выделения нескольких элементов. Однако для этого визуально удобнее CheckedListBox.

Список (ListBox)

Список содержит прокручиваемый набор объектов. По умолчанию пользователь вправе выбрать один (или несколько) элементов, используя клавиатуру или мышь. Выбранные объекты выделяются.

Отображаемые элементы списка задаются свойством Items — объектом типа ListBox.ObjectCollection. В классе ListBox.ObjectCollection определён метод Add, позволяющий добавлять новые объекты в набор. В списке отображаются текстовые названия элементов, возвращаемые методом ToString каждого объекта. Свойство Sorted объекта ListBox обеспечивает автоматическую сортировку элементов. После этого можно использовать свойство Items объекта ListBox для обращения к отдельным объектам наподобие того, как это делают с массивом. Список также можно «заселить» элементами из источника данных — объекта, основанного на массиве.

В свойстве SelectedIndex хранится индекс выбранного в данный момент элемента. Если выбранных элементов нет, его значение равно –1. Свойство SelectedItem представляет фактический выбранный элемент (объект). Оно может быть равным null. При выборе какого-либо элемента списка выражение lstbox.SelectedItem эквивалентно выражению lstbox.Items[lstbox.SelectedIndex].

Следующие два выражения также равносильны: lstbox.Text и lstbox.SelectedItem.ToString().

По умолчанию в списке разрешается выбрать только один элемент. Однако вы вправе разрешить множественное выделение, задав свойству SelectionMode одно из значений перечисления SelectionMode: None, One, MultiSimple или MultiExtended. Вариант MultiSimple слегка меняет вид ListBox, делая различия между выбранными элементами (указаны выделением) и фокусом ввода (указан пунктиром). Фокус ввода перемещают клавишами-стрелками, а пробелом или мышью выбирают или отменяют выбор элементов. Вариант MultiExtended разрешает выбирать только диапазон последовательных элементов, для чего пользователь должен, придерживая клавишу Shift, стрелками выбрать нужный диапазон.

В списках с возможностью выбора нескольких элементов используется свойство «только для чтения» SelectedIndices (объект типа ListBox.SelectedIndexCollection) или SelectedItems (объект типа ListBox.SelectedObjectCollection). Объекты ListBox.SelectedIndexCollection и ListBox.SelectedObjectCollection поддерживают индексы и доступ к отдельным элементам по механизму массива целых чисел.

Для получения информации об изменении выбранного элемента (или элементов) устанавливают обработчик события OnSelectedIndexChanged или OnSelectedValueChanged.

Если нужно, чтобы элементы списка отображались методом, отличным от ToString, можно использовать функцию элемента управления ListBox, называемую *отрисовка владельцем* (*owner draw*). В этом случае программа уведомляется посредством событий о необходимости отрисовки элемента списка.

При использовании отрисовки владельцем первым делом надо присвоить свойству DrawMode значение DrawMode.OwnerDrawFixed (все пункты списка имеют одну высоту) или DrawMode.OwnerDrawVariable (пункты списка разной высоты). По умолчанию значение свойства DrawMode равно DrawMode.Normal, что возлагает обязанность отрисовки на сам элемент управления.

Если все элементы списка одинаковой высоты, её нужно указать в свойстве ItemHeight. В противном случае необходимо задействовать событие Measurement, обработчик которого задаётся в соответствии с делегатом MeasureItemEventHandler. Предоставляемый сообщением объект MeasureItemEventArgs имеет свойства для чтения Graphics и Index. Первое представляет собой объект типа Graphics, служащий для вычисления высоты элемента, а последнее — числовой индекс элемента списка. Высота и ширина элемента задается свойствами ItemHeight и ItemWidth.

При отрисовке владельцем необходимо установить обработчик события DrawItem. Обработчик вызывается для каждого отображаемого элемента, при этом ему передаётся объект типа DrawItemEventArgs с несколькими свойствами ListBox «только для чтения»: BackColor, ForeColor и Font. Свойство Bounds — это объект типа Rectangle, в котором выполняется отрисовка. Кроме этого, обработчик получает объекты Graphics и Index. Свойство State — это член перечисления DrawItemState, которое указывает, выбран ли элемент, имеет фокус ввода и т. д. Класс DrawItemEventArgs также включает два метода, DrawBackground и DrawFocusRectangle, помогающие в отрисовке элемента списка.

Список с флажками (CheckedListBox)

При необходимости выбора нескольких элементов списка вместо ListBox лучше использовать CheckedListBox. В этом варианте слева от элементов расположены флажки; при этом интерфейсы взаимодействия с помощью клавиатуры и мыши проще и понятнее для пользователя, чем в списках с возможностью выбора нескольких элементов.

В списках с флажками SelectionMode может принимать только значение SelectionMode.One или SelectionMode.None. (В последнем варианте поведение элемента управления довольно неочевидно, поэтому его лучше не использовать.) В каждый момент времени выбран и выделен цветом только один элемент, однако флажками могут быть отмечены несколько пунктов. Можно сделать так, чтобы флаговые окна выглядели «притопленными», задав свойству ThreeDCheckBoxes значение true.

Перемещение фокуса выполняется клавишами со стрелками, а пробел служит для переключения флажка.

Первый щелчок мыши выделяет элемент списка, а следующий — устанавливает флажок. При выборе другого элемента состояние флажка останется неизменным. Можно избавиться от необходимости два раза щёлкать мышью, для этого нужно задать свойству CheckOnClick значение true. Теперь состояние флажка будет меняться одним щелчком.

Два прилагающихся к набору элементов метода Add позволяют при добавлении новых элементов автоматически устанавливать флажки. Состояние флажка определяется значением второго аргумента — true или false.

```
ckdlstbox.Add("New Jersey", true);
```

Для определения состояния элемента управления можно использовать встречавшееся ранее при описании элементов управления CheckBox перечисление CheckState, содержащее члены Checked, Unchecked или Indeterminate:

```
ckdlstbox.Add("New York", CheckState.Indeterminate);
```

Позже можно изменить состояние флажка, указав индекс элемента:

```
SetItemChecked(5, true);
```

Перечисление CheckState можно использовать при перегрузке метода:

```
SetItemCheckState(7, CheckState.Unchecked);
```

Свойства «только для чтения» CheckedIndices и CheckedItems предоставляют наборы целочисленных индексов или объекты, выбранные в данный момент. Чтобы получать информацию об изменении состояния флажка, нужно установить обработчик события ItemCheck в соответствии с делегатом ItemCheckEventHandler. Предоставляемый вместе с сообщением о событии объект ItemCheckEventArgs содержит свойство Index элемента, а также значения CurrentValue и NewValue, выраженные членами перечисления CheckState.

Поле со списком (ComboBox)

Поле со списком ComboBox — это комбинация списка и текстового окна. В обычном состоянии отображается только текстовое окно со стрелкой «вниз» справа. По щелчку этой стрелки раскрывается список — внизу появляется окно со списком. По умолчанию пользователь вправе ввести текст в текстовое окно (чтобы выбрать элемент списка), редактировать отображаемый в окне текст или ввести совершенно новый текст. Введённый текст не добавляется автоматически в список — это делается из программы. Большинство программ не разрешают редактирование текста. Такое поведение управляется свойством DropDownStyle — членом перечисления ComboBoxStyle, содержащего следующие члены:

- Simple — текст можно редактировать, список постоянно открыт;
- DropDown — текст можно редактировать, список отображается по щелчку (по умолчанию);
- DropDownList — текст редактировать нельзя, список отображается по щелчку.

Для раскрытия списка (или определения его положения в текущий момент) можно использовать свойство DroppedDown.

Как и в ListBox, в программе можно задействовать свойства SelectedIndex и SelectedItem для определения или получения выбранного элемента. Поля со списком не поддерживают множественный выбор. Выбранный элемент отображается в текстовом окне элемента управления и доступен через свойство Text. В момент ввода текста в поле текстового окна свойство SelectedIndex равно –1, а SelectedItem — null.

Как и в ListBox, для обнаружения момента изменения выбранного элемента задействуют обработчики событий SelectedIndexChanged или SelectedValueChanged. Событие TextChanged информирует об изменении текста в верхней части ComboBox — из-за выбора другого элемента или ввода текста.

Как и ListBox, ComboBox поддерживает отрисовку владельцем.

Абстрактный класс наборного счетчика (UpDownBase)

Наборные счётчики (*spin controls*) содержат как поле ввода, так и кнопки. Элемент управления NumericUpDown позволяет выбрать число из диапазона. DomainUpDown похож на ListBox тем, что позволяет выбрать один объект из набора. В обоих есть кнопки со стрелками, и оба реагируют на клавиши со стрелками.

Числовой наборный счетчик (NumericUpDown)

Самые важные свойства NumericUpdown — Minimum, Maximum, Increment и Value — всё это десятичные величины. (Десятичный тип в C# соответствует структуре Decimal пространства имён System. В отличие от float и double, десятичный тип данных позволяет хранить числа с точностью до 28 разрядов.) DecimalPlaces — это еще одно важное свойство NumericUpDown. Элемент управления можно инициализировать так:

```
NumericUpDown updn = new NumericUpDown();
updn.DecimalPlaces = 2;
updn.Minimum = 5.25m;
updn.Maximum = 5.50m;
updn.Increment = 0.05m;
```

Затем по нажатию клавиш или кнопок со стрелками свойство Value может принимать значения 5,25, 5,30, 5,35, 5,40, 5,45 и 5,50. Однако пользователю можно предоставить возможность вводить и промежуточные, лежащие между Minimum и Maximum, значения — например, 5,33. В этом случае при нажатии клавиши со стрелкой вверх следующее значение будет равно 5,38.

Два булевых свойства элемента управления по умолчанию равны false. Это ThousandsSeparator, которое добавляет в больших числах разделитель (в зависимости от региональных стандартов это может быть запятая или пробел), и Hexadecimal, которое отображает результаты в шестнадцатеричной системе.

При попытке ввода данных, выходящих за пределы заданного диапазона, прозвучит сигнал, и значение Value станет равным Minimum или Maximum. Значение false свойства Readonly запрещает ввод данных пользователем.

Наиболее важным событием элемента управления NumericUpDown является ValueChanged.

Счётчик выбора из диапазона (DomainUpDown)

Элемент управления DomainUpDown похож на NumericUpDown, но хранит объекты и отображает текстовые строки. Свойство Items — это набор объектов, хранящихся в экземпляре класса DomainUpDown.DomainUpDownItemCollection. Элементы в набор добавляются в точности, как в ListBox:

```
domain.Items.Add("New Jersey");
domain.Items.Add("New York");
domain.Items.Add("New Hampshire");
domain.Items.Add("New Mexico");
domain.Items.Add("New Rochelle");
```

Об изменении выбранного элемента информирует событие SelectedltemChanged. Доступ к выбранному элементу получают, используя свойство Selectedltem или выбрав элемент из набора Items по его индексу, указанному в свойстве Selectedlndex.

Дата и время

В .NET Framework определена структура DateTime в пространстве имён System, которая широко используется для хранения и отображения дат и времени. Как правило новый объект DateTime создаётся одним из двух способов: с помощью статического свойства Now, создающего объект DateTime с текущими датой и временем, или с помощью одного из множества конструкторов, позволяющих задать дату, со временем или без. Вот пример создания объекта DateTime, представляющего дату 29 августа 2006 г.:

```
DateTime dt = new DateTime(2006, 8, 29);
```

У структуры DateTime есть свойства «только для чтения» — Year, Month, Day, Hour, Minute, Second и Millisecond, служащие для получения информации о дате и времени, а также несколько методов для форматирования даты и времени в соответствии с региональными стандартами.

В Windows Forms есть два элемента управления для работы с датами. MonthCalendar предназначен для получения информации о дате от пользователя, а DateTimePicker позволяет получать ещё и время, но чаще всего используется только для ввода даты.

Календарь на месяц (MonthCalendar)

Элемент управления MonthCalendar отображает календарь с текущей датой, выделенной красным кружком и отображаемой внизу элемента управления. С помощью мыши или клавиатуры пользователь может выбрать до семи последовательных дней. Для этого надо выбрать первую, а затем, удерживая клавишу Shift, выделить последнюю дату диапазона. Для просмотра другого месяца нужно щёлкнуть одну из стрелок, расположенных вверху элемента управления, или нажать клавишу Page Up или Page Down.

В программе узнать выбранный диапазон позволяют два свойства типа DateTime: SelectionStart и SelectionEnd. (В качестве альтернативы можно использовать свойство SelectionRange — объект типа SelectionRange со свойствами Start и End.)

Чтобы получать уведомление об изменении выбранных дат, устанавливают обработчик события DateChanged, основанный на DateRangeEventHandler. В сообщении о событии содержится объект типа DataRangeEventArgs с двумя свойствами типа DateTime — Start и End. (В классе MonthCalendar реализовано еще одно событие — DateSelected, но оно практически бесполезно.)

Для увеличения/уменьшения количества последовательных дней, которые пользователь может выбрать в элементе управления, используется свойство MaxSelectionCount. Чтобы ограничить выбор одним днём, задайте свойству значение 1. Обычно MonthCalendar позволяет перемещаться сколь угодно далеко по времени (на нескольких столетий или даже тысячелетий). Для ограничения диапазона дат служат свойства MinDate и MaxDate типа DateTime.

Можно запретить отображение текущей даты внизу элемента управления, задав свойству ShowToday значение false. Чтобы запретить выделение текущей даты красным кружком, задайте свойству ShowTodayCircle значение false. Установка текущей даты выполняется присвоением соответствующего значения свойству TodayDate. Свойство ShowWeekNumbers предлагает нумерацию недель, отображаемую слева от календаря, начиная с первой недели года.

Если региональные стандарты требуют отображать календарь иначе, не нужно менять значение по умолчанию Day.Default свойства FirstDayOfWeek. Это позволит календарю корректно отображать названия дней недели.

Свойства BoldedDates, MonthlyBoldedDates и AnnuallyBoldedDates являются массивами объектов DateTime. Класс MonthCalendar содержит методы Add и Remove, используемые для создания и обслуживания этих массивов.

Элемент управления MonthCalendar может изменять свой размер. По умолчанию его ширина составляет 13-кратную, а высота — 11-кратную высоту шрифта. Можно изменить значение свойства Size, но нужно иметь в виду, что элемент управления проигнорирует изменения, если новые размеры недостаточны для размещения, по крайней мере, одного месяца. Если значение свойства Size таково, что в элементе управления можно разместить несколько месяцев (горизонтально, вертикально или в обоих направлениях), MonthCalendar отобразит их, но скорректирует свойство Size так, чтобы в элементе управления умещалось именно это количество месяцев.

Чтобы задать отображение заданного количества месяцев, во время исполнения проверьте свойство «только для чтения» SingleMonthSize. Оно учитывает текущий шрифт и свойства ShowToday и ShowWeekNumbers. Общая ширина должна быть несколько больше числа, кратного SingleMonthSize.Width, поскольку в элементе управления месяцы разделяются небольшим промежутком. (Возможно, надёжнее и проще просто добавить еще одну половину SingleMonthSize.Width.) Высота должна быть чуть меньше числа, кратного SingleMonthSize.Height, потому что размер одного месяца рассчитан на размещение текущей даты, а она одна на все месяцы и отображается только внизу элемента управления.

Например, для отображения шести месяцев — трёх в ширину и двух в высоту, нужно выражение:

```
moncal.Size = new Size(7 * moncal.SingleMonthSize.Width / 2,
    2 * moncal.SingleMonthSize.Height);
```

Если надо узнать точный размер элемента управления, запросите значение свойства Size.

Элемент выбора даты и времени (DateTimePicker)

Хотя у DateTimePicker больше возможностей, чем у MonthCalendar, так как он позволяет выбирать дату и время, его обычно используют только для определения даты (впрочем, это и есть режим по умолчанию). В отличие от MonthCalendar, у DateTimePicker нет возможности выбора диапазона дат.

С первого взгляда DateTimePicker походит на поле со списком с текстовым полем и стрелкой справа. При инициализации элемента управления из программы нужно задать свойству Value значение объекта типа DateTime, в противном случае ему будет присвоено значение DateTime.Now.

Следующим после Value по важности свойством элемента DateTimePicker является Format, которому присваивается значение одного из членов перечисления DateTimePickerFormat. По умолчанию это Long, при выборе которого дата отображается в формате «Tuesday, August 02, 2005» (в региональном стандарте U.S. English). Если выбрать DateTimePickerFormat.Short дата отобразится в формате «8/1/2005», а при выборе DateTimePickerFormat.Time элемент управления будет содержать только время. Чтобы определить пользовательский формат отображения даты и времени, свойству Format задают значение DateTimePickerFormat.Custom, а формат даты и времени определяют в свойстве CustomFormat.

Независимо от формата пользователь сможет напрямую редактировать поля (месяц, день, год и т. д.). Щелчок правой стрелки раскрывает календарь — так же, как и в MonthCalendar.

Событие ValueChanged информирует об изменении свойства Value.

Древовидное и списковое представление

Элементы управления TreeView и ListView являются основными компонентами Проводника Windows. Элемент управления TreeView отображает иерархический список и часто используется для отображения дерева папок или аналогичной структуры. ListView отображает список элементов в одном из нескольких форматов, в числе которых таблица, содержащая подробную информацию об элементах. В проводнике элемент ListView отображает файлы и подпапки папки, выбранной в TreeView. Оба элемента управления напрямую наследуются от Control.

Древовидное представление (TreeView)

Все элементы иерархического списка, отображаемые в TreeView, называются узлами и являются объектами типа TreeNode. Каждый узел может быть родителем других узлов. Потомки узла указаны в свойстве Nodes класса TreeNode, которое является объектом типа TreeNodeCollection — набора других объектов TreeNode. В TreeView также есть свойство Nodes, содержащее информацию об узлах верхнего уровня.

Объект TreeNode можно создать с помощью не получающего параметры конструктора, задав лишь значение свойству Text:

```
TreeNode nodeCats = new TreeNode();
nodeCats.Text = "Cats";
```

Также можно передать отображаемый в узле текст непосредственно конструктору, принимающему один аргумент:

```
TreeNode nodeSiamese = new TreeNode("Siamese");
```

Затем один из узлов делают потомком другого, используя метод Add узла TreeNodeCollection, доступный через свойство Nodes:

```
nodeCats.Nodes.Add(nodeSiamese);
```

Также можно создать узел неявно, передав текст узла другой версии метода Add:

```
nodeCats.Nodes.Add("Calico");
```

Конечно, для отображения всех этих узлов нужно создать объект TreeView:

```
TreeView tree = new TreeView();
```

К элементу управления TreeView можно добавить узлы верхнего уровня, используя те же методы Add:

```
tree.Nodes.Add(nodeCats);
tree.Nodes.Add("Dogs");
```

Позже можно получить доступ к этим узлам, обращаясь к ним по индексу с использованием свойства Nodes. Например, такое выражение возвращает созданный ранее узел Calico:

```
TreeNode node = tree.Nodes[0].Nodes[1];
```

В TreeNode есть много свойств, призванных облегчить навигацию по узлам. Среди них Parent, родительский узел, а также FirstNode, LastNode, NextNode и PrevNode, все из которых возвращают «родственные» узлы, то есть расположенные на одном уровне иерархии с текущим узлом. Еще одно интересное свойство TreeNode называется FullPath, оно возвращает текстовую строку, состоящую из конкатенации значений свойств Text всех родительских узлов вплоть до верха иерархии, разделенных обратным слэшем. Обратный слэш можно заменить другим знаком — он задаётся значением свойства PathSeparator.

В каждый момент времени в дереве элемента TreeView может быть выбран только один узел. Узлы могут быть развёрнутыми (при этом отображаются дочерние узлы) и свёрнутыми. Узел, содержащий дочерние узлы, помечен знаком «плюс» (+) в свернутом и знаком «минус» (–) — в развёрнутом состоянии. (Для управления отображением этих символов и линий, соединяющих узлы, служат свойства ShowPlusMinus, ShowLines и ShowRootLines.)

Развёртыванием/свёртыванием узлов можно управлять из программы, используя методы Expand, ExpandAll, Collapse и Toggle. У TreeNode есть также булевы свойства IsExpanded и IsSelected, информирующие о том, развернут или выбран ли узел соответственно. Свойство SelectedNode элемента TreeView позволяет выяснить, какой узел выбран, а также выбрать определённый узел.

Кроме этого существует целый набор событий, сигнализирующих о происходящем с узлом. При развёртывании/свёртывании и изменении выбранного узла инициируется пара событий: BeforeExpand и AfterExpand, BeforeCollapse и AfterCollapse или BeforeSelect и AfterSelect. Сообщения о событиях BeforeXXX содержат объект типа TreeViewCancelEventArgs. В обработчике событий можно отменить операцию, задав свойству Cancel значение true. Если Cancel оставить равным false, событие AfterXXX будет сопровождаться объектом типа TreeViewEventArgs. У TreeViewCancelEventArgs и TreeViewEventArgs есть, соответственно, свойства: Node, указывающее на задействованный в операции узел, и Action, являющееся членом перечисления TreeViewAction. Свойство Action предоставляет информацию о типе операции (развёртывание или свёртывание) и инструменте действия (клавиатура или мышь).

Обычно узлы TreeView отмечаются небольшими значками. При выборе папки в проводнике её значок меняется с «закрытого» на «открытый».

Изображения, отображаемые элементом управления TreeView, заданы в свойстве ImageList. Свойства ImageIndex и SelectedImageIndex элемента TreeView указывают на изображения по умолчанию для выбранных и невыбранных узлов. (Кроме этого, можно ссылаться на изображения набора ImageList по имени, используя ImageKey и SelectedImageKey) В классе TreeNode также есть свойства ImageIndex, SelectedImageIndex, ImageKey и SelectedImageKey, используемые для обозначения индивидуальных значков для отдельных узлов. Эти свойства всегда ссылаются на ImageList элемента TreeView, которому принадлежат эти узлы.

У TreeView есть и другие интересные возможности. Для отображения флажков рядом с узлами надо задать свойству Checkboxes значение true. В таком виде TreeView похож на CheckedListBox, но содержит иерархическое дерево, а не простой список.

Булево свойство Checked элемента TreeNode указывает, отмечен ли данный узел флажком. С этой возможностью связаны свойства StateImageList в TreeView и StateImageIndex со StateImageKey в TreeNode.

Чтобы пользователи могли редактировать метки, прежде всего, задайте свойству LabelEdit элемента TreeView значение true. Далее, нужно определить, как пользователь будет активизировать режим редактирования. Обычно используется щелчок правой кнопки мыши, для этого нужно установить обработчик события

NodeMouseClick, в котором убедиться, что значение свойства Button аргумента равно MouseButtons.Right, и затем вызвать для этого узла метод BeginEdit.

Затем TreeView генерирует событие BeforeLabelEdit. Объект NodeLabelEditEventArgs этого события имеет три свойства: Node, Label и CancelEdit. Свойство «только для чтения» Node — это редактируемый объект-узел TreeNode. Свойство Label равно null. Редактирование можно отметить, задав свойству CancelEdit значение true. В противном случае вы получите сообщение AfterLabelEdit с другим объектом NodeLabelEventArgs. Возможно, потребуется обработка этого события. Свойство Node хранит состояние узла, а свойство Text объекта — текст до редактирования. Свойство Label содержит новый текст. (Если пользователь отменит редактирование, нажав клавишу Esc либо щёлкнув в другом месте экрана, свойство Label станет равным null.) На этом этапе проверяется правомочность редактирования. Например, проводник не позволит задать пустую строку в качестве имени каталога, а также предупредит о возможных последствиях при изменении расширения файла. Чтобы запретить изменение, надо задать CancelEdit значение true. Иначе свойству Text узла будет присвоено значение свойства Label.

Списковое представление (ListView)

В проводнике предоставляется несколько вариантов отображения списка файлов: только имена файлов со значками или подробная информация о файлах, включая размер, тип, дату изменения и другие параметры. Эти способы отображения связаны с View, одним из самых важных свойств ListView. Оно принимает значения одноименного перечисления, состоящего из членов Details, List, Largelcon, Smalllcon и Tile.

Один из первых шагов настройки ListView — назначение колонок, отображающихся при выборе View.Details. Свойство Columns элемента ListView — это объект типа ColumnHeaderCollection, представляющего собой набор объектов ColumnHeader. Объекты ColumnHeader можно создавать по отдельности, затем, добавляя их к свойству Columns, или вы вправе воспользоваться методом Add набора ColumnHeaderCollection:

```
lstview.Columns.Add("File Name", 100, HorizontalAlignment.Left);
```

Первый аргумент определяет отображаемый в колонке текст, второй — ширину колонки, а последний аргумент задает выравнивание текста в заголовке и теле колонки — по левому или правому краю или по центру.

В режиме просмотра Details каждый ряд представлен объектом типа ListViewItem. Как минимум, надо задать свойство Text, отображаемое в первой колонке. Свойство SubItems объекта ListViewItem представляет вторую и каждую из последующих колонок. SubItems — это объект типа ListViewItem.ListViewSubItemCollection, представляющий собой набор объектов ListViewSubItem. Все относящиеся к ListView объекты ListViewItem собраны в свойстве Items.

Итак, элемент управления ListView — это набор объектов ListViewItem, каждый из которых в свою очередь содержит набор объектов ListViewSubItem. У ListView есть два свойства типа ImageList — SmalllmageList и LargelmageList, которые используются для хранения больших и маленьких значков.

Лекция 10. Динамическое размещение элементов. Панели и меню

Иногда случается так, что тщательно размещенные на форме или в диалоговом окне элементы странным образом перемешиваются на экране компьютера другого пользователя. Это может произойти из-за нестандартного разрешения экрана или странного шрифта по умолчанию, но, конечно же, элементы управления не должны накладываться друг на друга, а текст — не вмещаться на кнопках.

Разработчики программ для Windows традиционно размещали элементы управления на формах и в диалоговых окнах, жёстко задавая их координаты и размеры. Чтобы избежать проблем с интерфейсом из-за использования программ на системах с другим разрешением экрана и размерами шрифтов, применялись разные методы.

Программисты, использующие Microsoft Win32 или MFC (Microsoft Foundation Class), задают расположение и размеры элементов в особой аппаратно-независимой системе координат, в которой за основу взята одна восьмая высоты и одна четвёртая ширины системного шрифта Windows. Обычно эта система координат ограничена шаблонами диалоговых окон в сценариях ресурсов программы. При загрузке и отображении такого диалогового окна Windows преобразует эти единицы в пикселы, исходя из размеров текущего системного шрифта.

Но даже в аппаратно-независимой системе координат расположение всегда было малоприятной работой, поэтому Microsoft разработала одно из первых средств для «комфортного» программирования — конструктор форм, или визуальный редактор диалоговых окон. С его помощью программисты могли проектировать диалоговые окна интерактивно, перемещая и изменяя размеры элементов управления. Редактор также позволял создавать соответствующие сценарии ресурсов. Позже такие редакторы стали неотъемлемой частью интегрированных сред разработки, таких как Microsoft Visual Basic и Microsoft Visual Studio.

Изначально в Visual Studio существовало несколько способов размещения элементов с указанием их точных размеров и координат, однако, такие методы в последние годы становятся всё менее желательным. И на то есть ряд причин.

Во-первых, приложения переводятся на другие языки, и часто оказывается, что элемент управления, подогнанный по размерам к тексту на одном языке, не способен уместить текст на другом языке. Вместо перепроектирования формы и диалоговых окон, лучше сразу сделать так, чтобы элементы управления и диалоговые окна «автоматически» подстраивались к размерам текста на другом языке.

Во-вторых, мониторы с высоким разрешением со временем получают практически повсеместное распространение. Сегодняшние программы должны работать на системах с такими мониторами.

Третья проблема, возможно, имеющая наиболее серьёзные последствия, связана с самими элементами управления. Они становятся сложнее, и программистам становится труднее предугадать, сколько пространства потребуется для их размещения. Возможно, определение оптимального размера нужно делегировать самому элементу управления и выполняться это должно только во время выполнения. Форма или диалоговое окно, где находится этот элемент управления, должны изменять свои размеры в соответствии с его потребностями.

В прошлом формы определяли размеры элементов управления. В будущем, которое начинается с этого момента, элементы управления будут сами определять свои размеры, а формы — подстраиваться к ним.

Новая парадигма динамического размещения решает много проблем традиционного размещения. Во время выполнения программа размещает элементы управления, исходя из размера, затребованного этими элементами, разрешения, шрифта по умолчанию, размера окна программы и других факторов, известных только самим элементам управления.

Реализованное в классе `Control` и наследуемое всеми его потомками логическое свойство `AutoSize` — один из компонентов, позволяющих реализовать динамическое размещение в Windows Forms. По умолчанию `AutoSize` равно `false`, но если значение изменить на `true`, элемент управления станет менять свои размеры в зависимости от содержимого. Например, на кнопке всегда будет отображаться весь размещённый на ней текст.

Некоторые элементы управления дополнительно имеют свойство `AutoSizeMode`, принимающее значение одного из двух членов перечисления `AutoSizeMode`: `GrowAndShrink` или `GrowOnly` (по умолчанию).

В общем случае, при изменении размера элемента управления вызывается так называемый *диспетчер размещения*, который соответствующим образом сдвигает остальные элементы. Можно обрабатывать событие

Layout, реализованное в классе Control, и класс LayoutEngine из пространства имён System.Windows.Forms.Layout. Другой способ, который в большинстве случаев является более предпочтительным, — использование панелей FlowLayoutPanel и TableLayoutPanel, которые обеспечат динамическое размещение. В первой используется модель плавающего размещения, а в другой — табличная модель. Эти две модели вполне годятся для решения самого широкого диапазона задач по размещению.

Панели и контейнеры

У каждого элемента управления есть свойство Controls, которое может хранить набор других, дочерних элементов управления. Однако в большинстве элементов управления это свойство не используется — возможность иметь дочерние элементы управления важна только для таких элементов, как формы и панели.

Возможность поддерживать набор дочерних элементов управления — основная особенность класса ScrollableControl. Этот класс так называется потому, что может автоматически отображать полосы прокрутки, если размер не позволяет показать все дочерние элементы. Элемент управления, содержащий дочерние элементы, часто называют контейнером, несмотря на то, что класс ContainerControl является производным от ScrollableControl.

Элемент управления ContainerControl состоит из нескольких других элементов управления. Класс Form попадает в эту категорию, поскольку содержит клиентскую и неклиентскую области. Хотя элементы управления Panel производны от ScrollableControl, а не ContainerControl, они тоже считаются контейнерами, поскольку их основное предназначение — содержать другие элементы управления. Кроме этого, существуют элементы управления SplitContainer (и связанный с ним SplitterPanel), FlowLayoutPanel и TableLayoutPanel.

Стыковка

Свойство Dock реализовано в классе Control и наследуется всеми производными классами. Оно может принимать значение из перечисления DockStyle: None, Top, Bottom, Left, Right и Fill. У большинства элементов управления по умолчанию задано значение DockStyle.None.

Если, например, свойству Dock задать значение DockStyle.Top, элемент управления разместится в верхней части своего контейнера (например, в клиентской области формы) и займет всю его ширину от левого до правого края. Значение DockStyle.Top подходит для панелей инструментов, а DockStyle.Bottom — для строк состояния (именно такие значения по умолчанию назначаются элементам управления ToolBar и StatusBar, а также заменяющим их элементам ToolStrip и StatusStrip).

Обычно часть элементов управления размещается по краям формы, остальные — располагаются в центре. В последнем случае свойству Dock задают значение DockStyle.Fill.

Вот основные правила. Если задается значение свойству Dock одного элемента управления, нужно задать этому свойству всех элементов того же уровня значение отличное от DockStyle.None. Только один элемент управления может иметь свойство Dock равное DockStyle.Fill. Выполнение этих простых правил поможет избежать наложения элементов друг на друга. (Вместе с тем, если контейнер слишком маленький, стыкованные элементы все равно будут перекрывать друг друга — от этого никак не избавиться).

При задании различных (или одинаковых) значений свойству Dock разных элементов управления, нужно знать, как они будут взаимодействовать друг с другом.

При назначении элементам управлении свойства Parent (или при добавлении в родительский элемент управления методом Add свойства Controls), элементы управления выстраиваются в z-порядке (этот термин относится к третьей оси координат в трёхмерной системе координат). Z-порядок — это последовательность элементов управления в наборе элементов родителя. Элемент управления, расположенный в начале z-порядка, — обычно первый элемент, добавленный в набор. Ему соответствует индекс 0 родительского свойства Controls, и визуально он располагается над другими элементами, если с ними пересекается. Элемент управления в конце z-порядка — это последний элемент, добавленный в набор, на экране он располагается под остальными элементами. Z-порядок изменяют методами BringToFront и BringToBack.

Если несколько элементов одного уровня пристыкованы к одному краю контейнера, непосредственно к краю будет примыкать элемент управления, находящийся ближе всех к концу z-порядка, далее, ближе к центру располагается элемент, стоящий на ступеньку выше и так далее.

Поэтому верно простое правило: начинать нужно с создания центральных элементов управления, задавая свойству Dock значение DockStyle.Fill. Затем создают остальные элементы, а заканчивать надо элементом, который должен располагаться у самого края контейнера.

Простые панели

Иногда нужно создать форму с меню, панелью инструментов и строкой состояния, но без большого элемента управления TextBox (или любого другого) в центре — вместо него требуется разместить группу элементов: метки, списки, наборные счётчики и другие.

Конечно, можно сделать так, чтобы одни элементы управления (такие как MenuStrip и ToolStrip) стыковались к краю, а остальные дочерние элементы — нет. Непристыкованные дочерние элементы можно явно разместить так, чтобы они отображались между панелью инструментов и строкой состояния. Но помните, что размещение выполняется относительно верхнего левого угла клиентской области. Часть клиентской области уже занята элементами управления MenuStrip и ToolStrip. Нужно позаботиться, чтобы элементы управления не перекрывали друг друга. Если же форма становится такой узкой, что меню или панель инструментов переходит на вторую строку, нужно немного сместить все элементы управления вниз.

Гораздо лучше создать в центре клиентской области простой, не выполняющий никаких функций элемент управления, задать его свойству Dock значение DockStyle.Fill и сделать его родителем всех остальных элементов управления небольшого размера (меток, списков, счётчиков и т. д.). Такой элемент управления называется *панелью*. Положение дочерних элементов панели задается относительно верхнего левого угла панели, а не формы.

Свойство AutoScroll есть у всех элементов управления, производных от класса ScrollableControl. В обычных формах оно применяется не часто, но оказывается очень кстати при использовании панели, на которой отображается много элементов управления (например, графических окон).

В обычном элементе управления Panel нужно явно размещать элементы управления (или использовать стыковку). Однако в качестве альтернативы можно использовать элементы управления FlowLayoutPanel или TableLayoutPanel, которые мы рассмотрим позже. Можно также задействовать элемент управления SplitContainer в качестве родителя других панелей. При планировании пользовательского интерфейса в Windows Forms надо мыслить в терминах иерархий панелей и элементов управления.

Привязка

Свойство Anchor схоже со свойством Dock в том, что оно связывает элемент управления с одним или несколькими краями контейнера. Однако это свойство не прикрепляет элемент управления к краю контейнера, а позволяет ему сохранять постоянную дистанцию от края.

Еще одно отличие: свойству Dock программы можно задать значение только одного члена перечисления DockStyle. Со свойством Anchor можно связывать несколько членов перечисления AnchorStyles, объединив их битовым оператором ИЛИ (|). Заметьте, что множественно число в имени AnchorStyles (в отличие от одного значения в DockStyle) предполагает использование нескольких членов.

Каждый член AnchorStyles представлен одним битом. В перечисление входят следующие члены (в скобках указаны их значения): None (0), Top (1), Bottom (2), Left (4) и Right (8).

По умолчанию свойство Anchor имеет значение AnchorStyles.Top | AnchorStyles.Left. Поэтому когда форма становится больше, расположенные на ней элементы остаются на прежнем месте относительно левого верхнего угла формы. Если задать свойству Anchor значение AnchorStyles.Bottom | AnchorStyles.Right, элемент управления будет сохранять определённую дистанцию от правого нижнего угла. Если задать значение AnchorStyles.None, элемент управления остаётся в форме при изменении её размеров. Если привязать элемент управления ко всем четырём сторонам, при изменении размера формы он будет оставаться на месте и изменять размер в соответствии с изменением формы.

Нельзя смешивать свойства Dock и Anchor. Если задать свойству Dock элемента управления значение, отличное от DockStyle.None, свойство Anchor моментально примет значение по умолчанию. Точно так же, при задании свойству Anchor значения, отличного от значения по умолчанию, свойство Dock принимает значение DockStyle.None.

Разделители

Разделитель (*splitter*) — это тонкая горизонтальная или вертикальная полоска, используемая для изменения относительных размеров двух областей экрана. Разделители используются в Проводнике Windows для отделения дерева просмотра в левой части от списка в правой части. В Visual Studio разделители отделяют окно редактора от окон Solution Explorer и Error List.

В .NET Framework 1.x разделитель Windows Forms представлялся элементом управления класса `Splitter`, который размещался между двумя другими элементами управления (обычно панелями, списками или деревьями просмотра) с соблюдением строгого порядка стыковки. Если порядок был неверным, разделитель мог появиться у края окна, а не между элементами управления.

`Splitter` остался в .NET Framework следующих версий, но в новых программах лучше применять `SplitContainer` — он намного проще в управлении. `SplitContainer` создаёт и отображает на своей поверхности две панели типа `SplitterPanel`, отделяя их тонким (толщиной в 4 пиксела) горизонтальным или вертикальным разделителем. При перемещении мышью этого разделителя, изменяются относительные размеры двух элементов `SplitterPanel`.

Три наиболее важные свойства элемента управления `SplitContainer` — это `Panel1` и `Panel2` (которые дают доступ к двум элементам управления `SplitPanel`, которые создает элемент `SplitContainer`), а также `Orientation`, которое является членом перечисления `Orientation` и определяет вид разделителя: вертикальная или горизонтальная полоска.

Можно создать собственные элементы управления `SplitterPanel`, но возможности их применения ограничены. Конструктор `SplitterPanel` требует аргумента `SplitContainer`, а свойства `Panel1` и `Panel2` элемента `SplitContainer` доступны только для чтения.

Когда окно расширяется или сужается, размеры обеих панелей пропорционально увеличиваются или уменьшаются. Это следствие того, что свойство `FixedPanel` имеет значение `FixedPanel.None`. Задайте свойству `FixedPanel` значение `FixedPanel.Panel1`, если размеры должны изменяться только у панели `Panel2` (возможно, пользователь ожидает этого исходя из своего предыдущего опыта), или задайте значение `FixedPanel.Panel2`, чтобы менялась только панель `Panel1`.

Можно создать более сложные разделители, добавив дополнительные элементы управления `SplitContainer` и задать их родителям значения `Panel1` и `Panel2` предыдущего элемента `SplitContainer`.

Свойства `Padding` и `Margin`

Свойства элементов управления Windows Forms `Padding` и `Margin` задают отступ — соответственно внутри и вне элемента управления.

Свойство `Padding` обеспечивает дополнительный отступ внутри элемента управления. При автоматическом изменении размера элемента управления учитывается и свойство `Padding`.

Свойство `Padding` — это объект типа `Padding`, который является структурой, определяющий два параметрических конструктора. Общий случай таков:

```
new Padding(iLeft, iTop, iRight, iBottom);
```

Все аргументы определяются в пикселах. Когда нужно добавить дополнительный отступ по сторонам элемента, можно использовать следующее сокращение:

```
new Padding(iAll);
```

В структуре `Padding` пять доступных для чтения и записи свойств: `Left`, `Top`, `Bottom` и `All` (которые равны –1, если все четыре стороны не одинаковы). Свойства `Horizontal` и `Vertical` доступны только для чтения. Свойство `Horizontal` является суммой `Left` и `Right`, а `Vertical` — суммой `Top` и `Bottom`.

Для контейнера, такого как форма или панель, дополнительный отступ добавляется вокруг краев контейнера. Пристыкованные элементы управления сдвинуты от края. Свойство `Margin`, которое также является структурой типа `Padding`, влияет на отступ между элементами управления.

Размещение в панели FlowLayoutPanel

Модель реализации панели FlowLayoutPanel напоминает размещение HTML, за тем единственным исключением, что вместо текста и изображений используются элементы управления. (Если нужен текст, используйте элемент управления Label, а если изображение — PictureBox.) Панель FlowLayoutPanel отвечает за последовательное размещение дочерних элементов управления слева направо (по умолчанию), а затем сверху вниз.

Направление, в котором FlowLayoutPanel размещает элементы управления, определяется свойством FlowDirection. Оно принимает значения из перечисления FlowDirection и по умолчанию имеет значение FlowDirection.LeftToRight. Другие возможные значения — TopDown, RightToLeft и BottomUp.

В режиме по умолчанию дочерние элементы управления размещаются на панели FlowLayoutPanel слева направо. Если очередной элемент управления не вмещается в доступное пространство, он переносится в следующую строку (или столбец, если свойство FlowDirection имеет значение TopDown или BottomUp). Это следствие того, что свойство WrapContents по умолчанию равно true.

Если свойство WrapContents равно false, нужно самостоятельно обрабатывать переходы на новую строку (или в новый столбец), и даже если используется значение WrapContents по умолчанию, иногда требуется начать новую строку (или столбец) определенным элементом управления. Если flow — объект типа FlowLayoutPanel, а ctrl — его дочерний объект, перенос этого элемента на новую строку или столбец выполняется так:

```
flow.SetFlowBreak(ctrl, true);
```

Метод GetFlowBreak возвращает логическое значение с аргументом элемента управления.

Стыковка и привязка в панели FlowLayoutPanel

Может показаться, что свойства Dock и Anchor не играют особой роли в управлении элементами управления в FlowLayoutPanel, но это не так. Свойства Dock и Anchor дочерних элементов управления на панели служат для их вертикального выравнивания (для горизонтального размещения) или горизонтального выравнивания (для вертикального размещения).

Если направление размещения горизонтальное, высота любой горизонтальной строки элементов управления определяется высотой самого высокого элемента этой строки. Все элементы управления выравниваются по верхнему краю ряда. Для любого элемента управления доступны четыре варианта выравнивания:

- по верхнему краю самого высокого элемента (по умолчанию);
- по нижнему краю самого высокого элемента;
- по центру самого высокого элемента;
- растяжение по вертикали до достижения одинаковой высоты с самым высоким элементом.

Выравниванием элемента управления по вертикали можно управлять через свойство Anchor:

- AnchorStyles.Top — выравнивание по верхнему краю;
- AnchorStyles.Bottom — выравнивание по нижнему краю;
- AnchorStyles.Top | AnchorStyles.Bottom — выравнивание по центру;
- AnchorStyles.None — растяжение по вертикали до достижения той же высоты;
- биты AnchorStyles.Left и AnchorStyles.Right игнорируются.

Некоторых из этих эффектов можно достичь при помощи свойства Dock: в этом случае любые значения свойства Anchor, отличные от значения по умолчанию, игнорируются, точнее переопределяются:

- DockStyle.Top — выравнивание по верхнему краю;
- DockStyle.Bottom — выравнивание по нижнему краю;
- DockStyle.Fill или DockStyle.Left или DockStyle.Right — растяжение по вертикали.

Поскольку Anchor предоставляет больше возможностей, для выравнивания по вертикали рекомендуется использовать именно его.

Когда направление размещения вертикальное (сверху вниз или снизу вверх), ширина каждого столбца элемента управления определяется самым широким элементом. Элементы управления обычно выравниваются по левому краю. Свойство Anchor позволяет изменить выравнивание относительно самого широкого элемента управления в столбце:

- AnchorStyles.Left — выравнивание по левому краю (по умолчанию);
- AnchorStyles.Right — выравнивание по правому краю;
- AnchorStyles.Left | AnchorStyles.Right — выравнивание по центру;
- AnchorStyles.None — растяжение по горизонтали до достижения той же ширины;
- биты AnchorStyles.Top и AnchorStyles.Bottom игнорируются.

Однако FlowLayoutPanel не подходит для форм, содержащих элементы управления различных типов и размеров, которые нужно выстраивать как по горизонтали, так и по вертикали. Для этого лучше подойдет элемент управления TableLayoutPanel.

Панели TableLayoutPanel

Панель TableLayoutPanel размещает дочерние элементы в сетке строк и столбцов. Обычно каждый дочерний элемент управления занимает одну ячейку сетки, но ничто не запрещает размещать отдельные элементы на нескольких смежных клетках. Конечно, другие панели, содержащие другие элементы TableLayoutPanel, также могут занимать эти ячейки. Однако поскольку поддержка и управление таблицами довольно ресурсоемкое занятие, лучше не увлекаться вложением таблиц. Во многих случаях элемент FlowLayoutPanel справляется с подобной задачей не хуже, чем дочерние элементы TableLayoutPanel.

Есть два основных способа использования TableLayoutPanel. В большинстве случаев значением true инициируется свойство AutoSize элемента TableLayoutPanel и контейнера панели. Таким образом, размеры таблицы и её контейнера подгоняются для размещения элементов управления.

В качестве альтернативы можно предоставить пользователям возможность изменять размер контейнера таблицы либо в виде перемещаемой границы, либо с использованием элемента SplitContainer. В этом случае свойству Dock элемента TableLayoutPanel присваивается значение DockStyle.Fill. Документация предупреждает, что это не самый эффективный способ использования TableLayoutPanel, но это идеальный вариант для выполнения некоторых нестандартных задач.

Автоматическое расширение таблицы

Допустим, в конструкторе некоторого контейнера — Form или Panel — мы решили создать элемент TableLayoutPanel примерно так:

```
TableLayoutPanel table = new TableLayoutPanel();
table.Parent = this;
table.AutoSize = true;
```

Таблица обычно размещается в левом верхнем углу родительского элемента, но в качестве родителя может выступать FlowLayoutPanel или другой элемент TableLayoutPanel. Если свойству AutoSize не присвоить значение true, размер таблицы не будет автоматически увеличиваться для нормального отображения всех ячеек и придется явно задавать размер в пикселах.

Экспериментируя с TableLayoutPanel, лучше всего сделать границы (обрамление) ячеек видимыми:

```
table.CellBorderStyle = TableLayoutPanelCellBorderStyle.Single;
```

Доступны и другие стили обрамления (Inset, InsetDouble, Outset, OutsetDouble и OutsetPartial), но для большинства задач размещения обрамление вообще не понадобится.

Следующий шаг — создание элемента управления (например, Button), который нужно разместить в таблице:

```
Button btn = new Button();
btn.Text = "Button 1";
btn.AutoSize = true;
```

Как обычно, свойство AutoSize позволяет кнопке динамически менять размер в соответствии с размером текста.

Сделать кнопку потомком таблицы можно двумя способами:

```
btn.Parent = table;
```

или

```
table.Controls.Add(btn);
```

Если создать и добавить в таблицу ещё несколько кнопок с текстом, получится таблица с одним столбцом. При добавлении элементов в таблице появляются дополнительные ячейки. По умолчанию ячейки располагаются в один столбец. Такое поведение определяется тремя свойствами элемента TableLayoutPanel: RowCount, ColumnCount и GrowStyle.

По умолчанию значения свойств RowCount и ColumnCount равны 0, а GrowStyle принимает значение из перечисления TableLayoutPanelGrowStyle.AddRows. Свойства RowCount и ColumnCount НЕ говорят о том, сколько строк и столбцов имеет таблица. Элемент TableLayoutPanel НЕ изменяет начальные значения этих свойств или значения, назначенные программой.

Так как свойство GrowStyle по умолчанию имеет значение TableLayoutPanelGrowStyle.AddRows, свойство RowCount игнорируется. Свойству ColumnCount можно задать значение, определяющее нужное число столбцов. Значение 0 имеет тот же эффект, что и значение 1. Если свойству ColumnCount задать значение 2, таблица будет иметь две строки, и т.д. Значение свойства ColumnCount можно изменять в любое время при добавлении новых или реорганизации уже имеющихся элементов управления.

Если свойство GrowStyle имеет значение TableLayoutPanelGrowStyle.AddColumns, свойство ColumnCount игнорируется. Свойству RowCount можно присвоить значение, соответствующее нужному числу строк. Если значение свойства RowCount равно 0, элементы управления выстраиваются так же, как и при значении по умолчанию, а при 1 — выстраиваются горизонтально в одну строку. При бо́льших значениях RowCount размещение элементов управления изменяется при добавлении дополнительных элементов с учётом того, что RowCount определяет максимальное количество строк.

Третье возможное значение GrowStyle — TableLayoutPanelGrowStyle.FixedSize. Здесь число строк и столбцов фиксируется свойствами RowCount и ColumnCount. (Перед заданием свойству GrowStyle значения FixedSize нужно присвоить этим двум свойствам ненулевые значения.)

Если свойство GrowStyle имеет значение FixedSize, ячейки заполняются слева направо, а затем сверху вниз, как и при значении TableLayoutPanelGrowStyle.AddRows. Ситуация коренным образом меняется, когда число элементов управления превышает фиксированное число ячеек (значение RowCount умноженное на значение ColumnCount), — в элементе TableLayoutPanel возникнет исключение.

Позиции ячеек

При определённых обстоятельствах — например, при добавлении элементов управления, свойство GrowStyle которых имеет значение AddColumns, или при помещении элемента управления в ячейку, где уже имеется другой элемент — TableLayoutPanel перегруппирует элементы управления в ячейках таблицы. Механизм динамического размещения элементов управления основан на шести методах TableLayoutPanel, отвечающих за размещение ячеек.

В методах GetCellPosition и SetCellPosition используется структура TableLayoutPanelCellPosition, которая имеет два свойства — Column и Row. Можно использовать эти два метода по отношению к любому элементу управления, потомку TableLayoutPanel. В следующем примере lstbox — элемент управления класса ListBox, а table — класса TableLayoutPanel. Допустим, элемент lstbox добавляется в набор элементов управления таблицы методом Add с координатами столбца и строки:

```
table.Controls.Add(lstbox, col, row);
```

Затем программа вызывает метод GetCellPosition:

```
TableLayoutPanelCellPosition cellpos = table.GetCellPosition(lstbox);
```

Как и следовало ожидать, свойства Column и Row объекта cellpos будут указывать те же значения столбца и строки. Если объект lstbox добавлен в набор элементов управления таблицы простым методом Add или заданием значения свойству Parent, то вызвав метод GetCellPosition, можно выяснить, что значения cellpos.Column и cellpos.Row равны –1, иначе говоря, у элемента управления нет явно заданных координат ячеек. Если нужно, таблица может перемещать элементы управления в другие ячейки, то есть элемент управления может «плавать».

Методы GetColumn и GetRow полностью соответствуют GetCellPosisition, но они возвращают целочисленные значения. Поэтому, скорее всего, проще в использовании.

Предположим, в ячейку последовательно добавляются два элемента управления:

```
table.Controls.Add(lstbox, 3, 2);
table.Controls.Add(btn, 3, 2);
```

Второй метод Add «выталкивает» элемент ListBox в другую ячейку. Но при передаче этих элементов методу GetCellPosition (или GetColumn, или GetRow) обнаруживается, что они оба находятся в столбце 3 и строке 2, поскольку программа задала именно такие значения. Эти два элемента имеют больший приоритет при размещении в этой ячейке, чем элементы со значениями –1, но все же нет никаких гарантий, что они будут размещены именно в этой конкретной ячейке.

Область действия методов SetCellPosition, SetColumn и SetRow элемента управления TableLayoutPanel ограничена только имеющимися потомками таблицы. Можно использовать эти методы для перемещения элементов в другие ячейки. Если значения строки и столбца равны –1, элемент управления перемещается в первую пустую ячейку.

Если нужно узнать, в какой ячейке *действительно* находится конкретный элемент управления, используйте метод GetPositionFromControl, который также возвращает объект типа TableLayoutCellPosition:

```
TableLayoutPanelCellPosition cellpos = table.GetPositionFromControl(lstbox);
```

Теперь значения cellpos.Column и cellpos.Row показывают ячейку, в которой находится элемент управления. Вот еще один метод элемента управления TableLayoutPanel:

```
Control Ctrl = table.GetControlFromPosition(col, row);
```

Если в ячейке нет элемента управления, метод вернёт null.

Стили строк и столбцов

По умолчанию ширина ячейки равна ширине элемента управления, расположенного в ней. Но поскольку у всех ячеек в определенном столбце одна ширина, в действительности ширина ячейки равна ширине самого широкого элемента в этом столбце. Аналогично, высота ячейки равна высоте самого высокого элемента в строке. У таблицы есть также другие параметры, известные как *стили* строк и столбцов.

Свойства ColumnStyles и RowStyles элемента управления TableLayoutPanel являются наборами типов TableLayoutColumnStyleCollection и TableLayoutRowStylesCollection. В эти наборы входят объекты ColumnStyle и RowStyle соответственно.

Классы ColumnStyle и RowStyle производные от класса TableLayoutStyle. У последнего есть свойство SizeType. В класс ColumnStyle добавлено свойство Width, а в класс RowStyle — Height.

Свойство SizeType принимает значения из перечисления SizeType с тремя членами: AutoSize (по умолчанию), Absolute и Percent.

Не следует смешивать RowStyle и ColumnStyle, если размер строк и столбцов должен изменяться автоматически. Не нужно задавать значение RowStyle для всех строк таблицы — достаточно задать вплоть до последней строки, размер которой не нуждается в автоматическом изменении.

Значение SizeType.AutoSize указывает на то, что ширина столбца (или высота строки) должна определяться по самому большому элементу столбца (или строки). При этом неважно, равно или нет true

свойство AutoSize элементов управления в этом столбце (или строке). Свойство Width свойства ColumnStyle (или свойство Height свойства RowStyle) игнорируется.

Если задано значение SizeType.Absolute, ширина столбца (или высота строки) указывается в пикселах и определяется значением свойства Width (или Height).

Когда элемент TableLayoutPanel задаёт ширину и высоту столбцов и строк, в первую очередь обрабатываются столбцы и строки со стилями SizeType.Absolute и SizeType.AutoSize. Оставшееся пространство распределяется между столбцами и строками со стилем SizeType.Percent. Числовые значения Width и Height, связанные со стилем SizeType.Percent не имеют значения — важны только их относительные пропорции.

Элемент TableLayoutPanel «распределяет оставшееся пространство» среди ячеек со стилем SizeType.Percent. Это предполагает, что стиль SizeType.Percent не работает в таблицах, свойство AutoSize которых равно true — он годится для таблиц с явно определёнными размерами или со свойством Dock, имеющим значение DockStyle.Fill.

После определения размеров всех столбцов и строк в элементе TableLayoutPanel, в программе можно получить размеры в пикселах в виде массивов целых чисел:

```
int[] aiWidths = table.GetColumnWidths();
int[] aiHeights = table.GetRowHeights();
```

Когда нужно узнать номера столбцов и строк таблицы, наверно, проще всего воспользоваться следующими методами:

```
iNumCols = table.GetColumnWidths().Length;
iNumRows = table.GetRowHeights().Length;
```

Свойства Dock и Anchor

Для выравнивания элемента управления в ячейке можно воспользоваться свойствами Dock и Anchor элемента управления. Dock можно игнорировать, поскольку все, что это свойство позволяет делать, можно выполнить при помощи Anchor, и даже больше.

Свойству Anchor можно задать значение одного или нескольких членов перечисления AnchorStyles, разделённых битовым оператором ИЛИ (|). Как было описано ранее, в AnchorStyles входят следующие члены: Left, Top, Right, Bottom и None. Включение любого члена перечисления AnchorStyles в Anchor пристыковывает элемент управления к соответствующему краю ячейки, в противном случае элемент располагается в центре ячейки. Размер элемента управления не изменяется, если элемент не должен стыковаться к противоположным краями ячейки.

Диапазоны столбцов и строк

На практике редко бывает так, чтобы форму можно было втиснуть в таблицу с фрксированным количеством строк и столбцов. В большинстве случаев формы должны гибче работать с элементами управления, занимающими несколько ячеек таблицы.

Для этого используются *диапазоны* (*spans*) строк и столбцов, например:

```
table.SetColumnSpan(ctrl, 2);
```

Элемент управления ctrl должен быть потомком table. Второй аргумент задаёт диапазон в две ячейки. Если перед этим вызовом элемент ctrl занимает ячейку (2, 4), после него он будет занимать ячейки (2, 4) и (3, 4). Если в ячейке (3, 4) расположен другой элемент управления, он будет перемещён в соседнюю ячейку, возможно, сдвигая другие элементы управления и вызывая исключение, если ячеек недостаточно, чтобы вместить все элементы. В общем случае, лучше не полагаться на логику сдвига ячеек, если элементы управления должны располагаться в определённом порядке. Если у ячеек есть обрамление, граница между ячейками (2, 4) и (3, 4) не будет удалена.

Следующий элемент управления будет обрабатываться аналогично:

```
table.SetRowSpan(ctrl, 3);
```

Если перед этим вызовом элемент `ctrl` занимает ячейку (2, 4), после вызова он будет занимать ячейки (2, 4), (2, 5) и (2, 6).

Методы `SetColumnSpan` и `SetRowSpan` можно вызывать для одного и того же элемента управления. Методы `GetColumnSpan` и `GetRowSpan` принимают в качестве аргумента элемент управления и возвращают целочисленное значение диапазона.

Для элемента управления, занимающего несколько строк и столбцов, метод `GetPositionFromControl` возвращает самую верхнюю и самую левую ячейку, которую занимает элемент управления. Метод `GetControlFromPosition` возвращает один и тот же элемент управления для всех ячеек, которые он занимает.

Меню и панели инструментов

Было время, когда меню и панели инструментов легко различались. Меню представляли собой иерархические наборы текстовых элементов, а панели инструментов состояли из обычных кнопок с изображениями-растрами. Но когда панели инструментов стали поддерживать выпадающие меню, а меню стали содержать графику, различать их стало сложнее. Возможности меню и панелей инструментов стали практически одинаковыми.

`ToolStrip`, `MenuStrip` и `StatusStrip` являются классами, соответственно, для панелей инструментов, меню и строк состояния.

Элемент управления `ToolStrip` предоставляет большинство гибких современных функций панелей инструментов, присутствующих в сложных приложениях, таких как Microsoft Office. `ToolStrip` поддерживает не только кнопки с растрами, но и поля редактирования текста, поля со списками и выпадающие меню. Можно разместить в окне несколько таких элементов управления и позволить пользователям перемещать их и прикреплять к любому краю окна, или перемещать элементы с одной панели `ToolStrip` на другую.

Элемент управления `MenuStrip` предоставляет все возможности, доступные в `ToolStrip`, включая комбинирование текста и графики. Единственное существенное различие между `ToolStrip` и `MenuStrip` в том, что `MenuStrip` активизируется по нажатию клавиши `Alt`.

Если в программе используется класс `MainMenu`, клиентская область уменьшается, так как часть пространства занимает меню. А `MenuStrip` является элементом управления, расположенным в клиентской области (обычно в её верхней части), и если надо разместить там другие элементы управления, нужно учитывать пространство, занятое им (а также элементами управления `ToolStrip` и `StatusStrip`). В большинстве случаев панель размещается в центре клиентской области путём присваивания свойству `Dock` значения `DockStyle.Fill`.

Меню, как панели инструментов и строки состояния, обычно содержат много «элементов», или команд. Каждый элемент обычно обозначается кратким текстом. Некоторые элементы отображаются всегда — они называются элементами меню верхнего уровня, их можно выбирать при помощи клавиатуры или мыши. Обычно выбор элемента меню верхнего уровня приводит к появлению подменю или выпадающего меню, содержащего дополнительные элементы, часто представленные текстовыми строками, которые иногда дополняются изображениями и быстрыми клавишами. Элементы меню могут вызывать команды или открывать другие подменю. Подменю можно вкладывать друг в друга, причем глубина ограничена лишь чувством целесообразности программиста. Меню часто содержат горизонтальные линии, называемые *разделителями* (*separator*), которые помогают объединять связанные команды одного подменю.

Класс `MenuStrip` — это элемент управления, значение свойства `Dock` которого равно `DockStyle.Top`, и где отображаются элементы меню верхнего уровня, выстроенные в окне по горизонтали. Каждый элемент меню является отдельным объектом. В простых меню чаще всего используются элементы меню, основанные на классах с (не очень удачными) именами `ToolStripMenuItem` и `ToolStripSeparator`. Это не элементы управления, а экземпляры класса `Component` и абстрактного класса `ToolStripItem`.

Класс `MenuStrip` наследует от `ToolStrip` свойство `Items` типа `ToolStripItemCollection`, которое является набором объектов `ToolStripItem`. Набор `Items` содержит элементы меню верхнего уровня — обычно объекты `ToolStripMenuItem`.

Класс ToolStripMenuItem происходит от класса ToolStripDropDownItem. В этом классе реализовано свойство DropDownItems, также типа ToolStripItemCollection. Таким образом, у каждого из элементов верхнего уровня есть свойство DropDownItems, содержащее элементы подменю. Дальнейшее вложение выполняется за счет использования свойств DropDownItems этих элементов.

Элементы меню

Класс ToolStripItem (от которого происходит класс ToolStripMenuItem) не является производным от класса Control, но в нём реализованы некоторые относящиеся к элементам управления свойства и события. Например, в этом классе есть свойства Size, Width и Height (Location нет, но имеется доступное только для чтения свойство Bounds). Как и элементы управления, у объектов класса ToolStripItem есть свойства Margin, Paddling, Dock и Anchor.

Класс ToolStripItem также содержит свойства Font, ForeColor и BackColor. Например, если нужно назначить определённому элементу меню (или панели инструментов) шрифт Times New Roman размером 24 пункта, это можно сделать так:

```
item.Font = new Font("Times New Roman", 24);
```

Не менее часто применяются логические (булевы) свойства Enabled и Visible. Если свойство Enabled равно false, элемент становится блеклым и недоступным для пользователя. Если значение false присвоить свойству Visible, элемент станет невидимым.

Класс ToolStripItem позволяет отображать текстовую строку (определённую в свойстве Text) и/или изображение (из свойства Image). При использовании класса ToolStripMenuItem изображение всегда отображается слева от текста в отдельном столбце раскрывающегося меню (панели инструментов предоставляют более широкие возможности использования изображений).

Кроме задания значения свойству Image напрямую, программа может сопоставлять свойство ImageList набору изображений объекта ImageList, а затем ссылаться на эти изображения по свойствам ImageIndex и ImageKey класса ToolStripItem.

Поскольку класс Image в .NET Framework является родителем классов Bitmap и Metafile, поместить метафайл в меню (или на панель инструментов) так же просто, как и изображение. Изображения можно загружать из файлов и ресурсов или создавать их непосредственно в программе.

Класс ToolStripMenuItem расширяет класс ToolStripItem новыми возможностями. Быстрые клавиши определяются в свойстве ShortCutKeys с использованием одного или нескольких членов перечисления Keys, соединённых побитовым оператором «или». Например, так:

```
item.ShortCutKeys = Keys.Control | Keys.O;
```

Обычно создаётся соответствующая текстовая строка (в данном случае — Ctrl + O), но поведение можно изменить, определив в свойстве ShortcutKeyDisplayString другую строку или вообще запретив отображение быстрых клавиш путём присвоения свойству ShowShortcutKeys значения false.

Элементы меню могут также отмечаться флажками-галочками в зависимости от значений свойств Checked и CheckState. Свойство Checked булево, а свойству CheckState значения присваиваются из перечисления CheckState, состоящего из членов Checked, Unchecked и Indeterminate. Если свойство Checked элемента меню, с которым связано изображение, равно true, это изображение выделяется рамкой, говорящей о том, что элемент отмечен. Свойство CheckOnClick автоматически меняет состояние (установлен/сброшен) флажка по щелчку элемента меню.

Как и в Control, в классе ToolStripItem определено событие Paint и соответствующий метод OnPaint. Можно менять внешний вид элементов панелей инструментов или меню по собственному усмотрению, определяя производный от ToolStripItem класс с переопределёнными методами GetPreferredSize и OnPaint.

В классе ToolStripItem реализованы события мыши, но не клавиатуры. (Интерфейс клавиатуры для элементов панели инструментов или меню определяется классами ToolStrip или MenuStrip.) В частности, в классе ToolStripItem реализовано событие Click, являющееся основным способом оповещения программы о том, что пользователь инициировал выполнение команды из класса ToolStrip или MenuStrip.

Событие Click является одним из двух событий, которое часто встречается при работе с меню. Второе — DropDownOpening, реализованное в классе ToolStripDropDownItem и унаследованное от класса ToolStripMenuItem. Это событие инициируется перед отображением раскрывающегося меню и прекрасно подходит для включения или отключения элементов меню.

Сборка меню

Меню состоит из одного объекта MenuStrip и нескольких объектов ToolStripMenuItem, которые могут разделяться объектами ToolStripSeparator. Элементы верхнего уровня собираются в наборы Items объектов MenuStrip. Для многократного вложения в классе ToolStripMenuItem имеется свойство DropDownItems.

Создать объект MenuStrip и сделать его дочерним по отношению к форме можно так:

```
MenuStrip menu = new MenuStrip();
menu.Parent = this;
```

Значение свойства Dock равно DockStyle.Top.

Параметров сборки меню из отдельных элементов очень много. Для каждого элемента меню нужно создать (явно или неявно) объект типа ToolStripMenuItem:

```
ToolStripMenuItem itemOpen = new ToolStripMenuItem();
itemOpen.Text = "&Open...";
itemOpen.Image = bmOpen;
itemOpen.ShortcutKeys = Keys.Control | Keys.O;
itemOpen.Click += OpenOnClick;
```

Обратите внимание на знак амперсанда (&) в текстовой строке — он задает символ, который будет подчёркнут, когда пользователь нажмет клавишу Alt, позволяя ему выбрать команду меню с клавиатуры. Многоточие (...) общепринято для обозначения команды меню, вызывающей диалоговое окно. OpenOnClick — обработчик события.

У класса ToolStripMenuItem есть 5 других конструкторов, позволяющих определять текстовые строки, изображения, пары «текстовая строка + изображение», тройки «текстовая строка + изображение + обработчик события Click» или всё перечисленное и ещё быстрая клавиша:

```
ToolStripMenuItem itemOpen =
    new ToolStripMenuItem("&Open...",bmOpen, OpenOnClick, Keys.Control | Keys.O);
```

Кода меньше, но он не столь понятен. Можно также использовать конструктор, требующий только текстовую строку:

```
ToolStripMenuItem itemFile = new ToolStripMenuItem("&File");
```

Для элементов меню верхнего уровня можно также установить обработчик события DropDownOpening:

```
itemFile.DropDownOpening += FileOnDropDown;
```

Следующий код добляет команду Open (Открыть) в раскрывающееся меню File (Файл):

```
itemFile.DropDownItems.Add(itemOpen);
```

А так команда File вводится в основное меню MenuStrip:

```
menu.Items.Add(itemFile);
```

Эти два метода Add не обязательно вызывать в определённом порядке. Задавать значения свойств объекта ToolStripMenuItem можно и после его добавления в набор. Набор Items используется только для добавления в меню элементов верхнего уровня. Для добавления новых команд в меню верхнего уровня используют набор DropDownItems. Элементы меню отображаются в том порядке, в каком они добавлялись в набор.

Наборы Items и DropDownItems имеют тип ToolStripItemCollection. В этом классе реализованы дополнительные методы Add, позволяющие неявно создавать объекты ToolStripMenuItem. Вместо явного создания элемента File и добавления его в меню следующим образом:

```
ToolStripMenuItem itemFile = new ToolStripMenuItem("&File");
menu.Items.Add(itemFile);
```

можно выполнить это одним оператором:

```
ToolStripMenuItem itemFile = menu.Items.Add("&File");
```

Метод Add возвращает созданный объект, что удобно, если с ним нужно сделать что-то ещё, например добавить обработчик события DropDownOpening. Хотя в документации по методу Add сказано, что при добавлении элемента в MenuStrip возвращается объект типа ToolStripItem, на самом деле создается и возвращается объект ToolStripMenuItem. При всем желании невозможно создать объект ToolStripItem при помощи этого метода, так как это абстрактный класс.

Класс ToolStripItemCollection тоже включает метод Add, который позволяет определить изображение, «текст + изображение» или «текст + изображение + обработчик события Click». Более того, этот класс содержит метод AddRange, позволяющий добавлять несколько элементов за раз. Есть две версии этого метода: одна принимает аргумент типа ToolStripItemCollection, другая — массив объектов ToolStripItem. Последнюю версию метода можно использовать следующим образом:

```
menu.Items.AddRange(new ToolStripMenuItem[] { itemFile, itemEdit, itemView, itemHelp });
```

Есть ещё один конструктор ToolsStripMenuItem, который требует строку, изображение и массив объектов ToolStripItem и позволяет создавать и собирать раскрывающееся меню следующим образом:

```
itemFile = new ToolStripMenuItem("&File", null,
    new ToolStripMenuItem[] { itemOpen, itemSave, itemSaveAs, itemClose });
```

Поля и обращение к элементам меню

В программе может потребоваться ссылаться на некоторые элементы меню, и самый простой способ сделать это — хранить элементы как поля. Команды, которые должны становиться доступными только при определенных условиях, могут храниться как поля:

```
ToolStripMenuItem itemCut, itemCopy, itemPaste, itemDelete;
```

Кроме этого, нужно установить обработчик события DropDownOpening для раскрывающегося меню, в котором размещены эти команды:

```
itemEdit.DropDownOpening += EditOnDropDownOpening;
```

Метод EditOnDropDownOpening делает доступными или отключает элементы itemCut, itemCopy, itemPaste и itemDelete.

Даже если элементы меню не хранятся как поля, к ним можно обратиться через наборы Items и DropDownItems. Можно индексировать эти наборы и обращаться к ним за нужными элементами, как к массивами. Это может быть непросто, поскольку потребуется приведение типов. Наборы Items и DropDownItems — это объекты типа ToolStripItem, у которого нет свойства DropDownItems — оно реализовано в классе ToolStripDropDownItem, стоящего в иерархии над классом ToolStripMenuItem.

Допустим, объект menu типа MenuStrip сохранён как поле. Можно найти объект ToolStripMenuItem команды таким образом:

```
ToolStripMenuItem item =
    (ToolStripMenuItem)((ToolStripMenuItem)menu.Items[1]).DropDownItems[2];
```

Из набора Items в объекте menu можно извлечь элемент с заданным индексом. Затем этот элемент можно привести к объекту ToolStripMenuItem, чтобы получить доступ к набору DropDownItems. Аналогично, можно обратиться к другому элементу по его индексу, и таким же образом привести его к объекту типа ToolStripMenuItem.

Хотя приведение не очень удобно, но использование явных числовых индексов чревато бо́льшими неудобствами. Если добавить новые элементы меню или изменить их порядок, придётся менять индексы. Есть лучший способ: в документации по ToolStripItemCollection сказано, что есть альтернативный способ индексации элементов — по текстовым строкам, а именно по свойству Name. Для этого нужно определять свойство Name при создании каждого элемента меню:

```
ToolStripMenuItem itemEdit = new ToolStripMenuItem("&Edit");
itemEdit.Name = "Edit";
item = new ToolStripMenuItem("&Paste");
item.Name = "Paste";
```

Рекомендуется использовать свойство Name, которое совпадает или практически идентично тексту элемента меню, но без амперсандов и многоточий, например:

```
ToolStripMenuItem item =
    (ToolStripMenuItem)((ToolStripMenuItem)menu.Items["Edit"]).DropDownItems["Paste"];
```

Элементы управления, элементы меню и владельцы

На экране отображаются только элементы управления, т. е. экземпляры подклассов класса Control. Экземпляры классов MenuStrip, ToolStrip и StatusStrip являются элементами управления и обычно занимают прямоугольную область на экране.

Элемент управления MenuStrip отображает элементы-команды меню верхнего уровня, содержащиеся в наборе Items, но они не элементы управления. Пространство, занимаемое ими на экране, в действительности управляется объектом MenuStrip, к которому эти элементы принадлежат. Когда происходит перерисовка объекта MenuStrip (возможно, в результате восстановления окна программы после сворачивания), эти элементы перерисовывают сами себя, но на поверхности элемента управления MenuStrip.

При щелчке элемента меню верхнего уровня появляется прямоугольное раскрывающееся меню, где отображаются содержимое набора DropDownItems этого элемента. Поскольку это меню занимает пространство на экране, оно должно быть элементом управления. Однако элемент, который мы щёлкаем — это объект типа ToolStripMenuItem, который не является элементом управления. Элементы, отображаемые в раскрывающемся меню, также относятся к типу ToolStripMenuItem. Но как же создается элемент управления, содержащий элементы меню?

Класс ToolStripDropDownItem (от которого происходит класс ToolStripMenuItem) включает свойство DropDown типа ToolStripDropDown, восходящее к классу Control. Именно этот элемент управления служит для отображения раскрывающихся меню.

Каждый раз при создании объекта типа ToolStripMenuItem, тот создаёт элемент управления типа ToolStripDropDownMenu (который наследуется от класса ToolStripDropDown). Свойство Visible этого объекта ToolStripDropDownMenu изначально имеет значение false; этот объект хранится в свойстве DropDown объекта ToolStripMenuItem. Если щёлкнуть этот элемент, он проверит набор DropDownItems. Если последний непустой, элемент использует для отображения этих элементов на экране элемент управления, хранящийся в свойстве DropDown.

Классы ToolStripDropDown и ToolStripDropDownMenu используются для отображения «временных» (непостоянных) элементов в элементах управления ToolStrip и MenuStrip.

Все сказанное выше необходимо для понимания концепции «владельца» элементов меню. У класса ToolStripItem есть свойство Owner типа ToolStrip. Оно определяет элемент управления, на котором отображается данный элемент. У элементов меню верхнего уровня свойство Owner указывает на объект MenuStrip, на котором они отображаются. У любого другого объекта ToolStripMenuItem это свойство указывает на объект ToolStripDropDownMenu.

У ToolStripItem также есть защищённое свойство Parent. Обычно оно указывает на тот же объект, что и свойство Owner. Однако если на ToolStrip недостаточно места для отображения всех элементов, оставшиеся группируются в дополнительное раскрывающееся меню, на которое будет ссылаться свойство Parent. Однако свойство Owner не меняет своего значения.

У ToolStripDropDown есть ещё свойство OwnerItem типа ToolStripItem, указывающее на элемент, вызвавший раскрывающееся меню.

Если item — экземпляр класса ToolStripMenuItem, следующее выражение всегда равно true:

```
item == item.DropDown.OwnerItem
```

Вызываемое элементом свойство OwnerItem раскрывающегося меню всегда ссылается на этот элемент. Если itemFile и itemOpen — экземпляры класса ToolStripMenuItem, и элемент itemOpen добавлен в набор DropDownItems объекта itemFile, тогда следующее выражение всегда будет равно true:

```
itemFile.DropDown == itemOpen.Owner
```

Оба выражения ссылаются на объект ToolStripDropDownMenu, который вызывается из меню File и содержит команду Open. Это выражение требует некоторых вычислений, но также имеет значение true:

```
itemFile == ((ToolStripDropDownMenu)itemOpen.Owner).OwnerItem
```

Владельцем элемента Open является объект ToolStripDropDownMenu, на котором этот элемент отображается. Свойство OwnerItem при этом ссылается на элемент File, который вызывает раскрывающееся меню. Следующее выражение также равно true:

```
itemFile == (ToolStripDropDownMenu)itemFile.DropDownItems[0].Owner).OwnerItem
```

Установка и снятие флажков

Класс ToolStripDropDownMenu важен для работы с меню, хотя напрямую взаимодействовать с ним приходится редко. Он отвечает за отображение поля слева от элементов меню, которое (по умолчанию) используется для размещения изображений и флажков.

Для отображения флажка (галочки) рядом с командой меню, нужно задать свойству Checked значение true или свойству CheckState — одно из значений перечисления CheckState.Checked. Флажок снимается присвоением свойству Checked значения false или свойству CheckState — значения CheckState.Unchecked. Единственное преимущество CheckState перед Checked заключается в наличии параметра CheckState.Indeterminate, который позволяет вместо флажка отображать точку, которая информирует, что состояние флажка не определенно. Свойства Checked и CheckState можно изменять в обработчике события Click главного меню.

Если свойству CheckOnClick задать значение true, состояние флажка меняется по щелчку автоматически. Но если программа должна как-то реагировать на установку/сброс флажка, придётся установить обработчик события для этой команды меню.

Свойство CheckOnClick не очень подходит для выбора одного из нескольких взаимоисключающих элементов. В таких случаях почти всегда приходится использовать один обработчик события Click для всех элементов. Обычно этот обработчик снимает флажок с отмеченного элемента и устанавливает его на элементе, инициировавшем событие Click. Можно выполнить и другие действия, но они определяются логикой конкретных элементов меню.

Добавление изображений

Изображения рядом с пунктами меню полезны, когда аналогичное изображение располагается на панели инструментов. Пользуясь меню, пользователь привыкает к характерной «картинке» той или иной команды и, в конце концов, начинает пользоваться панелью инструментов, за счёт чего его работа становится эффективнее.

Изображения в программе проще всего использовать как *ресурсы*. Ресурсы — это двоичные данные, внедренные в файл `.exe` и легко доступные программе. Например, если проект содержит файл Paste.bmp, и это изображение отмечено как *внедрённый ресурс* (*Embedded Resource*), можно загрузить его в программу как объект `Bitmap`, используя следующий код:

```
Bitmap bmPaste = new Bitmap(GetType(), "namespace.Paste.bmp");
```

Первый передаваемый конструктору `Bitmap` аргумент должен ссылаться на какой-либо класс, определённый в программе. В пределах любого экземплярного метода или конструктора в любом определённом в программе классе можно просто использовать метод `GetType`. Второй аргумент — это имя файла изображения, которому предшествует пространство имён ресурсов программы (выделенное курсивом слово *namespace*). Не путайте пространство имён ресурсов с пространствами имен .NET Framework. Пространство имён ресурсов используется исключительно по отношению к ресурсам, и по умолчанию в Visual Studio оно совпадает с именем проекта. Изменить это значение можно в свойствах проекта.

Значение свойства `Image` элемента меню можно задать следующим образом:

```
itemPaste.Image = bmPaste;
```

Это всё, что нужно для отображения изображения в меню. Изображение будет отображаться слева от текста меню, в том же столбце, который используется для отображения флажков.

Другой способ использования изображений сложнее, но лучше подходит для управления большим числом изображений. Сначала создается объект `ImageList`:

```
ImageList imglst = new ImageList();
```

Каждое загружаемое изображение добавляется в список изображений по его ключевому имени:

```
imglst.Images.Add("Paste", new Bitmap(GetType(), "Paste.bmp"));
```

Сам список изображений привязывается к объекту `MenuStrip`:

```
menu.ImageList = imglst;
```

А ссылка на изображение из элемента меню выполняется по ранее назначенному ключевому имени:

```
itemPaste.ImageKey = "Paste";
```

Если планируется использовать много изображений, можно даже определить класс, производный от `ImageList`, и применить конструктор этого класса для загрузки всех нужных программе ресурсов-изображений.

Независимо от используемого способа, нужно также указать, какой цвет в изображениях должен имитировать прозрачность. Если используется объект `ImageList`, достаточно указать этот цвет только раз:

```
imglst.TransparentColor = Color.Magenta;
```

В противном случае вам придётся указывать цвет для каждого элемента меню, у которого есть изображение:

```
itemPaste.ImageTransparentColor = Color.Magenta;
```

Столбец в левой части раскрывающегося меню обычно используется для отображения как изображений, так и флажков. Поведение и внешний вид этого столбца управляется двумя свойствами класса `ToolStripDropDownMenu` — `ShowImageMargin`, которое по умолчанию равно `true`, и `ShowCheckMargin` (значение по умолчанию — `false`). В такой конфигурации по умолчанию для вывода изображений и флажков используется один столбец. Если у элемента меню есть и изображение, и флажок, последний не отображается, но изображение заключается в рамку.

Если свойству ShowImageMargin присвоить false, а ShowCheckMargin — true, будут видны только флажки, но не изображения. Если оба свойства равны false, скрыты будут и изображения, и флажки. Если же обоим свойствам присвоить значение true, будет отображаться два столбца — для изображений и для флажков.

Нестандартные элементы меню

Можно создать нестандартный (пользовательский) элемент меню как потомка класса ToolStripItem и переопределить, как минимум, методы GetPreferredSize и OnPaint. Однако есть способ получше. Можно поместить любой элемент управления на MenuStrip или ToolStrip, «инкапсулировав» его в класс ToolStripControlHost. Это производный от ToolStripItem класс, имеющий один конструктор, получающий на вход объект типа Control. Этот элемент управления ведёт себя как элемент меню или панели инструментов.

Когда нужно, чтобы производный от ToolStripControlHost класс вызывал некоторые события родителя, определяют открытое событие (возможно, с тем же именем, что и у родительского элемента управления) и его обработчик. Последний должен вызывать событие, определённое в родительском классе. В производном классе также переопределяют методы OnSubscribeControlEvents и OnUnsubscribeControlEvents, которые устанавливают и удаляют обработчик события. В этих методах также нужно вызывать методы OnSubscribeControlEvents и OnUnsubcribeControlEvents базового класса, чтобы обеспечить правильную установку других событий в классе ToolStripControlHost. Также необходимо переопределить обработчик OnClick, чтобы *явно* закрывать меню. Без этого раскрывающееся меню продолжает отображаться после выбора элемента щелчком мыши.

Контекстное меню

Класс ContextMenuStrip наследуется от класса ToolStripDropDownMenu. Этот элемент управления создаётся внутри программы и служит для отображения раскрывающихся меню. Контекстные меню обычно отображаются по щелчку правой кнопки мыши. Элементы меню ContextMenuStrip это обычно объекты ToolStripMenuItem, но вы вправе задействовать для этой цели любые объекты любого производного от ToolStripItem класса.

В сущности, есть два способа использования контекстных меню. В классе Control определено свойство ContextMenuStrip, которое можно инициализировать конкретным объектом ContextMenuStrip. По щелчку элемента управления правой кнопкой открывается контекстное меню.

Контекстное меню появляется по щелчку правой кнопкой любой части клиентской области формы. Если нужно получить больше контроля над отображением контекстного меню, можно установить обработчики события «щелчок правой кнопки мыши» и отображать контекстное меню методом Show, который ContextMenuStrip наследует от ToolStripDropDown. Аргументы метода Show позволяют указать место отображения контекстного меню.

Панели инструментов и их компоненты

Панели инструментов создаются и используются схожим с меню образом. Элементы меню с текстом могут также содержать изображения, вместе с тем, изображения расположены всегда слева от текста. Панели инструментов гибче в этом отношении. Если нужно отобразить текст и изображение, их взаимное расположение определяется свойством TextImageRelation, которому можно задать значение из одноимённого перечисления, содержащего члены TextAboveImage (текст над изображением), ImageAboveText (изображением над текстом), TextBeforeImage (текст поверх изображения), ImageBeforeText (изображение поверх текста, это значение по умолчанию) или Overlay (накладка).

Часто на панелях инструментов тексты очень короткие, а изображения — непонятны, в этом случае очень кстати оказываются всплывающие подсказки. Это короткие пояснения, появляющиеся во всплывающей рамке, когда курсор мыши задерживается над элементом. Обычно отображается текст из свойства Text, но это поведение можно изменить, задав значение свойству ToolTipText. Чтобы запретить появление всплывающих подсказок, задайте свойству AutoToolTip значение false.

Кнопки на панели инструментов

Существует три типа классов кнопок, производных от ToolStripItem. Наверно, чаще всего используется класс ToolStripButton, который обычно применяется для выполнения определенных команд. Как обычно, нужно установить обработчик события Click, чтобы отслеживать щелчки по кнопке. Рекомендуется использовать одни и те же обработчики событий Click для элементов меню и панелей инструментов.

Активизация и отключение объектов ToolStripButton немного отличается от включения и отключения элементов меню. Элементы раскрывающегося меню не нужно делать доступными для пользователя, пока меню не отображается. Однако, многие элементы панели инструментов видимы постоянно.

Элемент ToolStripButton может также функционировать в качестве выключателя. Как и у ToolStripMenuItem, у элемента ToolStripButton есть логическое свойство Checked и свойство CheckState, принимающее значения из перечисления CheckState. Если задать свойству CheckOnClick значение true, кнопка будет менять своё состояние при каждом её щелчке. Несколько элементов ToolStripButton можно использовать как группу переключателей, позволяющую выбрать один из нескольких возможных вариантов.

Если кнопка должна вызывать раскрывающийся список элементов, нужно использовать класс ToolStripDropDownButton. Как и ToolStripMenuItem, этот класс наследует свойства от ToolStripDropDownItem и имеет свойство DropDownItems, которое является набором объектов ToolStripItem. Возможно, устанавливать обработчик события Click не придётся, но обычно устанавливают обработчик события DropDownOpening, если нужно включать или отключать определённые элементы раскрывающегося меню.

Пользователи Microsoft Office, наверняка, привыкли как к кнопкам, выполняющим определенное действие, так и к кнопкам с маленьким треугольником, щелчок которых отображает раскрывающееся меню. Такой стиль кнопки реализован в классе ToolStripSplitButton, который также производный от ToolStripDropDownItem. Добавляют элементы в раскрывающееся меню такой кнопки так же, как и в ToolStripDropDownButton. Однако, устанавливать обработчик события Click нужно не всегда, поскольку это событие инициируется по щелчку любой части кнопки. Чтобы получать уведомления о щелчке части кнопки, не вызывающей раскрывающееся меню, нужно установить обработчик события ButtonClick.

Элементы управления как элементы ToolStrip

Помимо кнопок (и весьма тривиального элемента ToolStripLabel), есть другая крупная категория элементов, где можно размещать панели инструментов; она включает элементы ToolStripComboBox, ToolStripProgressBar и ToolStripTextBox. Все эти три класса происходят от класса ToolStripControlHost, и этот класс работает оболочкой элементов управления так, что они становятся пригодными для использования в меню и панелях инструментов. Как можно догадаться, классы ToolStripComboBox, ToolStripProgressBar и ToolStripTextBox позволяют использовать на панелях инструментов обычные элементы ComboBox, ProgressBar и TextBox. (Элемент ToolStripProgressBar чаще всего размещают в элементах управления StatusStrip, расположенных в нижней части окна.)

Создание нескольких панелей инструментов

Если необходимо создать несколько панелей инструментов, и нужно, чтобы все панели инструментов оставались в верхней части окна (или в любой другой его части), можно создать панель ToolStripPanel как потомка формы. Панель ToolStripPanel должна явно стыковаться к одному краю окна. Все объекты ToolStrip становятся дочерними для этой панели. Используйте метод Join класса ToolStripPanel, чтобы разместить объекты ToolStrip в нужных местах на панели. Пользователь сможет перемещать отдельные объекты ToolStrip в пределах общей панели.

На эту панель можно поместить и объекты MenuStrip (как в приложениях Microsoft Office), тогда пользователь сможет перемещать меню под панелью. Свойству GripStyle объекта MenuStrip можно задать значение ToolStripGripStyle.Visible — в крайней левой части объекта MenuStrip появится небольшая «ручка», за которую объект можно перетаскивать мышью.

Программа может быть еще гибче, предоставляя пользователю возможность перемещать элементы ToolStrip и MenuStrip прикреплять их к любому из четырех краев окна или, как минимум, к верхнему и нижнему. Для этого элемент управления ToolStripContainer должен быть потомком формы, а его свойство

Dock равняться DockStyle.Fill. ToolStripContainer разделяет поверхность формы на пять частей. Сверху, снизу, слева и справа располагаются четыре объекта ToolStripPanel, идентифицируемые свойствами TopToolStripPanel, LeftToolStripPanel, BottomToolStripPanel и RightToolStripPanel. Объекты ToolStrip и MenuStrip размещаются на этих панелях путем назначения их потомками объекта ToolStripPanel или использования метода Join этого объекта. Можно запретить использование любой из этих панелей при помощи свойства TopToolStripPanelVisible и аналогичных.

В центре этих четырех элементов управления ToolStripPanel располагается элемент управления ToolStripContentPanel, доступный через свойство ContentPanel. Здесь можно размещать другие элементы управления, которые должны присутствовать на форме.

Кроме этого, можно позволить пользователю изменять расположение отдельных элементов объекта ToolStrip, задав свойству AllowItemReorder значение true. Пользователь может перетаскивать мышью (удерживая кнопку Alt) отдельные элементы в пределах панели ToolStrip. В случае с несколькими панелями инструментов, пользователь может перетаскивать элементы с одной панели на другую, но только в том случае, если свойство AllowItemReorder обеих панелей равно true.

Программа должна сохранять пользовательскую настройку панели инструментов до следующего запуска программы. Для этого организуют обработку событий ItemRemoved и ItemAdded в каждой панели инструментов, где разрешено перемещение отдельных элементов. При перемещении элемента эти два события инициируются последовательно. (Эти события также инициируются, если сама программа добавляет или удаляет элемент панели инструментов.) Предоставляемый с событием объект представляет собой панель, с которой элемент удаляется или на которую размещается. Эти события предоставляются с объектами ToolStripItemEventArgs, у которых есть свойство Item, идентифицирующее перемещаемый элемент.

Строка состояния

После меню и панелей инструментов класс StatusStrip кажется простым. Строки состояния обычно располагаются в самом низу окна, поэтому свойство Dock по умолчанию равно DockStyle.Bottom. Если к нижней части окна пристыкованы другие элементы управления (например, ToolStripPanel) строку состояния нужно добавлять на форму после них.

Обычно в форме присутствует одна строка состояния, но ничто не мешает разместить несколько таких элементов управления. Обычно правый нижний угол объекта StatusStrip содержит треугольничек, «взявшись» за который пользователь может изменить размер формы. Эту возможность можно отключить, задав свойству SizingGrip значение false.

На панели ToolStripPanel можно также разместить один или несколько объектов StatusStrip. Если нужно разрешить пользователю перемещать эти объекты, задайте свойству GripStyle значение ToolStripGripStyle.Visible.

Набор элементов, отображающихся в строке состояния, обычно ограничиваются объектами ToolStripStatusLabel (которые просто отображают текст) и ToolStripProgressBar. Это единственные компоненты, которые Visual Studio позволяет разместить на строке состояния, но при желании можно разместить там и другие элементы.

Свойство AutoSize объекта ToolStripStatusLabel по умолчанию равно true, так что ширина метки зависит от её содержимого. По умолчанию у меток нет обрамления. Если есть несколько элементов ToolStripStatusLabel, отображаемый в них текст будет повторяться.

Можно инициировать AutoSize значением false — это позволит явно изменить свойство Width (ширина) метки. По умолчанию текст будет выравниваться по центру метки, но можно задать иное выравнивание, задав соответствующее значение свойству TextAlign. Однако лучше использовать свойство Margin. Размер метки будет подгоняться под объем содержимого, но с некоторым запасом экранного пространства. Хотя поля можно задать для всех четырёх сторон, лучше ограничиться левой и правой.

Еще одна популярная возможность — обрамление меток. Задайте свойству BorderSides одно или несколько значений из перечисления ToolStripStatusLabelBorderSides — None, Left, Top, Right, Bottom и All. Внешний вид обрамления задается свойством BorderStyle, которое принимает значение из перечисления Border3DStyle. Значение по умолчанию — Border3DStyle.Flat.

Часто приходится располагать одни метки слева, другие — справа от строки состояния (для этого существует свойство `Alignment`), а третьи — посередине. Если задать свойству `Spring` метки, расположенной посередине, значение `true`, то метка будет использовать всё оставшееся пространство, что даст искомый результат. Если же для всех элементов места недостаточно, текст, выходящий за пределы метки справа, виден не будет.