

Никита Кульгин

Основы программирования в *Microsoft Visual C# 2010*

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.068+800.92Visual C# 2010

ББК 32.973.26-018.1

К90

Кульгин Н. Б.

К90 Основы программирования в Microsoft Visual C# 2010. — СПб.:
БХВ-Петербург, 2011. — 368 с.: ил.+ CD-ROM — (Самоучитель)

ISBN 978-5-9775-0589-5

Книга является пособием для начинающих по программированию в Microsoft Visual C# 2010. В ней в доступной форме изложены принципы визуального проектирования и событийного программирования, на примерах показана технология создания программ различного назначения. Приведено описание среды разработки и базовых компонентов. Рассмотрены вопросы программирования графики, разработки программ работы с базами данных Microsoft Access и Microsoft SQL Server Compact Edition. Уделено внимание технологии LINQ, отладке программ, созданию справочной системы, установке созданной программы на компьютер пользователя. В справочнике приведено описание базовых компонентов и наиболее часто используемых функций. Прилагаемый компакт-диск содержит проекты, рассмотренные в книге.

Для начинающих программистов

УДК 681.3.068+800.92Visual C# 2010

ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Анна Кузьмина</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.12.10.

Формат 70×100¹/16. Печать офсетная. Усл. печ. л. 29,67.

Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

Оглавление

Предисловие.....	9
ЧАСТЬ I. MICROSOFT VISUAL C# 2010	11
Глава 1. Среда разработки Microsoft Visual C# 2010	13
Установка	13
Первый взгляд	14
Справочная информация	19
Глава 2. Первый проект	21
Начало работы над проектом	21
Форма.....	22
Компоненты.....	26
Событие	35
Функция обработки события.....	36
Структура проекта	40
Главный модуль	40
Модуль формы	42
Сохранение проекта.....	46
Компиляция	46
Ошибки	47
Предупреждения	48
Запуск программы.....	49
Исключения	49
Обработка исключения.....	50
Внесение изменений	54
Завершение работы над проектом	58
Установка приложения на другой компьютер.....	58
Глава 3. Базовые компоненты	61
<i>Label</i>	61
<i>TextBox</i>	64
<i>Button</i>	68
<i>CheckBox</i>	72

<i>RadioButton</i>	76
<i>GroupBox</i>	79
<i>ComboBox</i>	83
<i>PictureBox</i>	88
<i>ListBox</i>	94
<i>ListView</i>	98
<i>ImageList</i>	101
<i>ToolTip</i>	103
<i>Panel</i>	105
<i>CheckedListBox</i>	106
<i>Timer</i>	108
<i>NumericUpDown</i>	111
<i>StatusStrip</i>	115
<i>NotifyIcon</i>	117
<i>ToolStrip</i>	123
<i>MenuStrip</i>	125
<i>OpenFileDialog</i>	128
<i>SaveFileDialog</i>	130
ЧАСТЬ II. ПРАКТИКУМ ПРОГРАММИРОВАНИЯ.....	139
Глава 4. Графика.....	141
Графическая поверхность	143
Карандаши и кисти	144
Карандаш	144
Кисть	147
Графические примитивы	152
Линия.....	154
Ломаная линия.....	158
Прямоугольник.....	158
Точка	159
Многоугольник.....	160
Эллипс и окружность.....	160
Дуга	161
Сектор	162
Текст.....	167
Битовые образы.....	172
Анимация.....	174
Глава 5. Базы данных	185
База данных и СУБД.....	185
Локальные и удаленные базы данных	185
Структура базы данных	186
Компоненты доступа к данным	186
Создание базы данных	188
База данных Microsoft Access	188
Доступ к данным	188
Отображение данных	200

Выбор информации из базы данных	204
SQL-запрос	204
Работа с базой данных в режиме формы.....	207
Сервер баз данных Microsoft SQL Server Compact Edition	213
Среда Microsoft SQL Server Management Studio	213
Создание базы данных	213
База данных "Контакты".....	217
Развертывание приложения работы с базой данных Microsoft SQL Server Compact Edition	226
Глава 6. Консольное приложение	227
Создание консольного приложения	230
Запуск консольного приложения	235
Глава 7. LINQ	237
Лямбда-выражение	237
Q-оператор.....	237
Выполнение Q-оператора.....	239
Операции с массивами	240
Поиск в массиве	240
Обработка массива.....	241
Обработка массива записей.....	243
Работа с XML-документами.....	245
Отображение XML-документа	245
Глава 8. Отладка программы.....	253
Классификация ошибок.....	253
Предотвращение и обработка ошибок	255
Отладчик	258
Трассировка программы.....	258
Точки останова программы.....	258
Добавление точки останова	259
Удаление точки останова	259
Наблюдение значений переменных	260
Глава 9. Справочная информация	263
Справочная система HTML Help	263
Подготовка справочной информации	264
Основы HTML	265
Microsoft HTML Help Workshop	266
Файл проекта	267
Оглавление.....	269
Идентификаторы разделов	272
Компиляция	273
Доступ к файлу справочной информации.....	274
Отображение справочной информации	274
Глава 10. Публикация приложения	279
Подготовка к публикации.....	279
Мастер публикации.....	282
Установка	285

Глава 11. Примеры программ	287
Экзаменатор	287
Требования к программе	288
Файл теста.....	288
Форма	290
Доступ к файлу теста	292
Текст программы "Экзаменатор"	295
Запуск программы.....	303
Сапер.....	306
Правила и представление данных.....	307
Форма	308
Игровое поле	309
Начало игры.....	310
Игра	312
Справочная информация	316
Информация о программе	317
Текст программы	319
Глава 12. Краткий справочник	329
Форма.....	329
Компоненты.....	330
<i>Button</i>	330
<i>ComboBox</i>	331
<i>ContextMenuStrip</i>	332
<i>CheckBox</i>	333
<i>CheckedListBox</i>	334
<i>GroupBox</i>	334
<i>ImageList</i>	335
<i>Label</i>	335
<i>ListBox</i>	336
<i>MenuStrip</i>	337
<i>NotifyIcon</i>	337
<i>NumericUpDown</i>	338
<i>OpenFileDialog</i>	338
<i>Panel</i>	339
<i>PictureBox</i>	339
<i>RadioButton</i>	340
<i>ProgressBar</i>	341
<i>SaveFileDialog</i>	342
<i>TextBox</i>	343
<i>ToolTip</i>	344
<i>Timer</i>	344
Графика.....	344
Графические примитивы	344
Карандаш	346
Кисть	347
Типы данных	349
Целый тип	349
Вещественный тип	349
Символьный и строковый типы	349

Функции.....	349
Функции преобразования	349
Функции манипулирования строками	350
Функции манипулирования датами и временем.....	352
Функции манипулирования каталогами и файлами.....	353
Функции обработки имен файлов.....	355
Математические функции	355
События	356
Исключения	357
Приложение. Описание компакт-диска	359
Предметный указатель	361

Предисловие

Microsoft Visual Studio — популярнейший из инструментов разработки прикладных программ различного назначения. Одним из его компонентов является Microsoft Visual C# — среда разработки .NET-приложений. В нее на основе единого интерфейса интегрированы удобный конструктор форм, специализированный редактор кода, высокоскоростной оптимизирующий компилятор, отладчик, мастер развертывания и другие полезные инструменты.

Среда Microsoft Visual C# поддерживает технологию так называемой быстрой разработки (ее также часто называют "компонентной"). В ее основе лежит технология визуального проектирования и событийного программирования, суть которой заключается в том, что среда разработки берет на себя большую часть рутины, оставляя программисту работу по созданию диалоговых окон (визуальное проектирование) и функций обработки событий (событийное программирование).

Среда Microsoft Visual C# ориентирована на разработку .NET-приложений различного типа: Windows Forms, консольных, WPF (Windows Presentation Foundation), которые могут работать в операционных системах Windows, в том Windows Vista и Windows 7.

Технология Microsoft .NET основана на идее универсального программного кода, который может быть выполнен любым компьютером, вне зависимости от используемой операционной системы. Универсальность программного кода обеспечивается за счет предварительной (выполняемой на этапе разработки) компиляции исходной программы в универсальный промежуточный код (CIL-код, Common Intermediate Language), который во время запуска (загрузки) программы транслируется в выполняемый. Преобразование промежуточного кода в выполняемый осуществляется JIT-компилятор (от Just In Time — в тот же момент, "на лету"), являющийся элементом виртуальной выполняющей системы (Virtual Execution System, VES). Выполнение .NET-приложений в операционной системе Microsoft Windows обеспечивает Common Language Runtime (CLR, общезыковая исполняющая среда) — компонент Microsoft .NET Framework.

Усилия Microsoft по развитию и продвижению технологии .NET привели к тому, что в общем объеме вновь создаваемого программного обеспечения различного назначения увеличивается доля .NET-приложений, программ, ориентированных на платформу .NET. Это объясняется, прежде всего, возможностями, которые предос-

тавляет платформа прикладным программам, и тем, что технология .NET поддерживается новейшими операционными системами — начиная с Microsoft Windows Vista, платформа .NET является неотъемлемой частью операционной системы.

Чтобы понять, что такое .NET, какие возможности она предоставляет программисту, необходимо опробовать ее в деле. Для этого нужно изучить среду разработки, освоить технологию разработки, понять назначение и возможности компонентов, их свойства и методы.

Книга, которую вы держите в руках, посвящена практике программирования в Microsoft Visual C# 2010, разработке приложений Windows Forms (.NET). В ней, в объеме, необходимом для начинающего программиста, приведено описание среды разработки и базовых компонентов, раскрыта суть технологии визуального проектирования и событийного программирования, показан процесс создания программ различного назначения: от простейших, до программ работы с файлами, графикой и базами данных, уделено внимание технологии LINQ.

Состоит книга из двух частей.

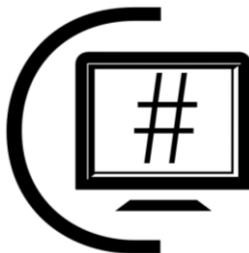
Часть I книги знакомит читателя с Microsoft Visual C++ 2010: содержит краткое описание среды разработки, демонстрирует процесс создания программы, назначение базовых компонентов.

Часть II посвящена практике. В ней рассмотрены задачи программирования графики, баз данных (Microsoft Access и Microsoft SQL Compact Edition), использования технологии LINQ для работы с массивами и XML-документами, создания справочной системы и развертывания приложений на основе технологии ClickOnce. Уделено внимание разработке консольных приложений. В главе-справочнике приведено описание базовых компонентов и наиболее часто используемых функций.

Цель этой книги — научить читателя программировать в среде Microsoft Visual C#, создавать программы различного назначения: от простых однооконных приложений до программ работы с базами данных. Следует обратить внимание на то, что хотя книга ориентирована на читателя, обладающего начальными знаниями и опытом в программировании, она вполне доступна и начинающим.

Научиться программировать можно только программируя. Поэтому, чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Изучайте листинги, старайтесь понять, что и как делают программы, как они работают. Не бойтесь экспериментировать — вносите в программы изменения. Чем больше вы сделаете самостоятельно, тем большему научитесь!

Visual

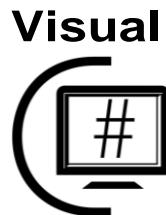


ЧАСТЬ I

Microsoft Visual C# 2010

В этой части книги приведено краткое описание среды разработки Microsoft Visual C# 2010; на примере программы "Конвертер" показан процесс разработки приложения Windows Forms; дано описание и приведены примеры использования базовых компонентов.

Глава 1.	Среда разработки Microsoft Visual C# 2010
Глава 2.	Первый проект
Глава 3.	Базовые компоненты



ГЛАВА 1

Среда разработки Microsoft Visual C# 2010

Установка

Среда разработки Microsoft Visual Studio 2010, в которую входит Microsoft Visual C#, доступна в трех вариантах: Professional, Premium и Ultimate. Каждый комплект представляет собой набор инструментов, обеспечивающих разработку высокоэффективных приложений для Win32- и .NET-платформ, Web-приложений. Различие вариантов состоит в составе компонентов — чем выше уровень (от Professional к Ultimate), тем больше возможностей и инструментов поддержки процесса разработки он предоставляет программисту. Так, например, в Ultimate включены инструменты UML-моделирования и анализа, в версии Premium есть только возможность просмотра UML-диаграмм, а в версии Professional работа с UML-диаграммами не поддерживается.

ЗАМЕЧАНИЕ

Помимо перечисленных выше коммерческих версий Microsoft Visual Studio 2010 корпорация Microsoft бесплатно предоставляет всем желающим Express-версию продукта. Отличие версии Express от коммерческих состоит в том, что она предназначена исключительно для целей обучения, изучения среды разработки и не дает право разработки коммерческих продуктов. Загрузить Microsoft Visual Studio 2010 Express целиком или только отдельные ее компоненты (например, Microsoft Visual C# 2010 Express) можно с узла <http://microsoft.com/express> (<http://www.microsoft.com/express/Downloads/#2010-Visual-CS>). Как и коммерческие версии Visual Studio, версия Express требует активации. Ключ активации предоставляется бесплатно. При этом следует учитывать, что лицензия на использование Express-версии именная, т. е. предоставляется конкретному физическому лицу.

Microsoft Visual Studio 2010 может работать в среде операционных систем Microsoft:

- ◆ Windows XP (x86) Service Pack 3 (кроме Starter Edition);
- ◆ Windows XP (x64) Service Pack 2 (кроме Starter Edition);
- ◆ Windows Vista (x86, x64) Service Pack 1 (кроме Starter Edition);
- ◆ Windows 7 (x86, x64);
- ◆ Windows Server 2003 (x86, x64) Service Pack 2;

- ◆ Windows Server 2003 R2 (*x86, x64*);
- ◆ Windows Server 2008 (*x86, x64*) Service Pack 2;
- ◆ Windows Server 2008 R2 (*x64*).

Особых требований, по современным меркам, к ресурсам компьютера Microsoft Visual Studio не предъявляет: процессор должен быть с частотой не ниже 1,6 ГГц, 1 Гбайт оперативной памяти (2 Гбайт для *x64*-процессора), порядка 7 Гбайт свободного места на жестком диске.

Установка выполняется с DVD-диска, на котором помимо Visual Studio находятся все компоненты, необходимые для установки и работы среды разработки (в том числе Microsoft .NET Framework 4).

При инсталляции с DVD процесс установки активизируется автоматически, после того как установочный диск будет помещен в дисковод (если в системе автоматический запуск программ запрещен, то установщик (файл setup.exe, который находится в корне установочного диска) надо запустить вручную).

Установщик проверяет версии Windows и Microsoft .NET Framework. Если операционная система, установленная на компьютере, не соответствует требуемой, процесс установки прерывается. В случае если версия .NET Framework ниже 4, то на компьютер устанавливаются компоненты Microsoft .NET Framework 4. После этого непосредственно начинается установка Microsoft Visual Studio.

Процесс установки Visual Studio обычный. Сначала на экране появляется окно лицензионного соглашения, затем установщик предлагает выбрать вариант установки: **Full** (Полный) или **Custom** (Выборочный). Выбрав вариант установки **Custom**, в следующем окне программист может указать компоненты Visual Studio, которые надо установить.

Первый взгляд

Microsoft Visual C# позволяет создавать .NET-приложения различного типа: Windows Forms, WPF (Windows Presentation Foundation), консольные (Console). Приложения Windows Forms — это обычные приложения .NET Windows. Приложения WPF для создания пользовательских интерфейсов используют новую, построенную на основе DirectX, графическую подсистему, обладающую расширенными возможностями (прозрачность, градиентная заливка, трансформация, отображение трехмерной графики, работа с растровой и векторной графикой, работа с аудио и видео, возможность связи интерфейсных элементов с источниками данных). Консольные приложения для взаимодействия с пользователем используют консоль — окно, обеспечивающее отображение текстовой информации, и клавиатуру.

Для того чтобы приступить к созданию программы в Microsoft Visual C# или, как принято говорить, начать работу над *проектом*, надо:

1. В меню **File** выбрать команду **New Project**.
2. В открывшемся окне **New Project** (рис. 1.1) выбрать тип приложения — **Windows Forms Application Visual C#**.
3. В поле **Name** ввести имя проекта и нажать кнопку **OK**.

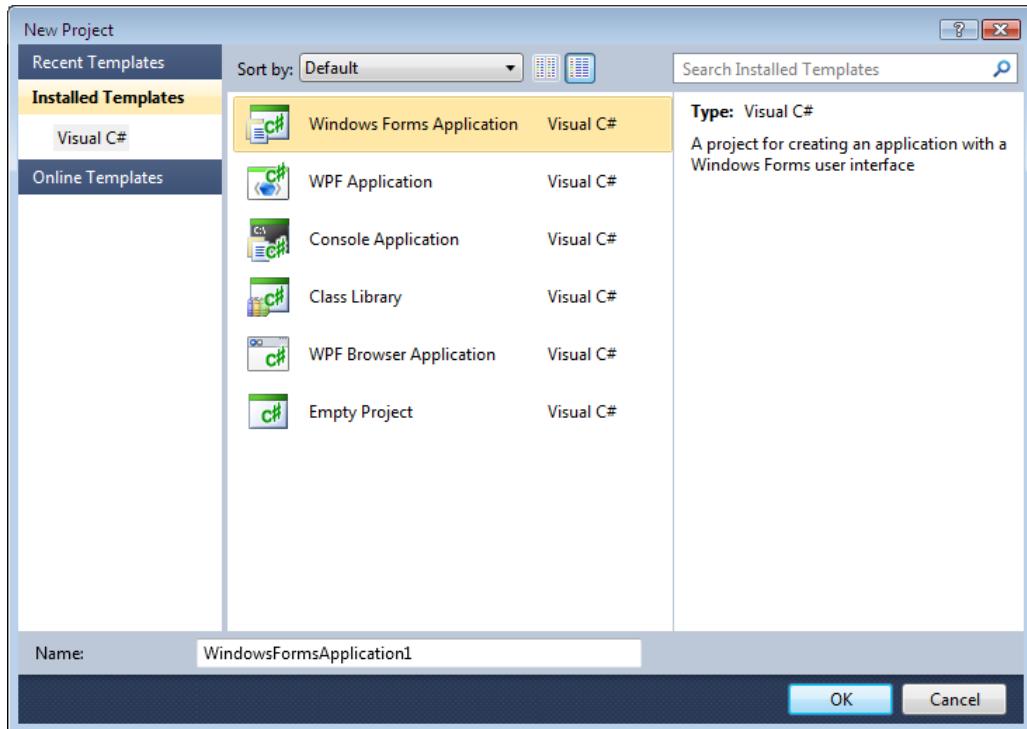


Рис. 1.1. В окне New Project надо указать тип приложения и задать имя проекта

В результате описанных действий будет создан *проект* — совокупность файлов, необходимых для создания компилятором выполняемого (exe) файла программы. Проект создается в папке временных проектов (по умолчанию это C:\Users\User\AppData\Local\Temporary Projects\Project, где: *User* — имя пользователя в системе; *Project* — имя проекта, указанное в момент его создания в окне New Project).

Главное окно Microsoft Visual C# в начале работы над новым проектом приведено на рис. 1.2. В его заголовке отображается имя проекта, над которым в данный момент идет работа.

В верхней части главного окна находится строка меню и область отображения панелей инструментов. По умолчанию в области панелей инструментов отображается панель **Standard** (рис. 1.3). Чтобы сделать доступными другие панели инструментов, надо в главном меню выбрать команду **View ▶ Toolbars** и в раскрывшемся списке сделать щелчок на имени нужной панели.

Центральную часть окна Visual C# занимает окно конструктора (Designer) формы (рис. 1.4). В нем находится *форма* — заготовка окна приложения (окно программы во время его разработки принято называть формой). В окне дизайнера в графической форме отображаются инструкции программы, обеспечивающие создание окна. Чтобы их увидеть (рис. 1.5), в окне **Solution Explorer** раскройте список **Form1.cs**, сделайте двойной щелчок на элементе **Form1.Designer.cs** и в открывшемся окне раскройте область **Windows Form Designer generated code** (код, сгенерированный дизайнером формы).

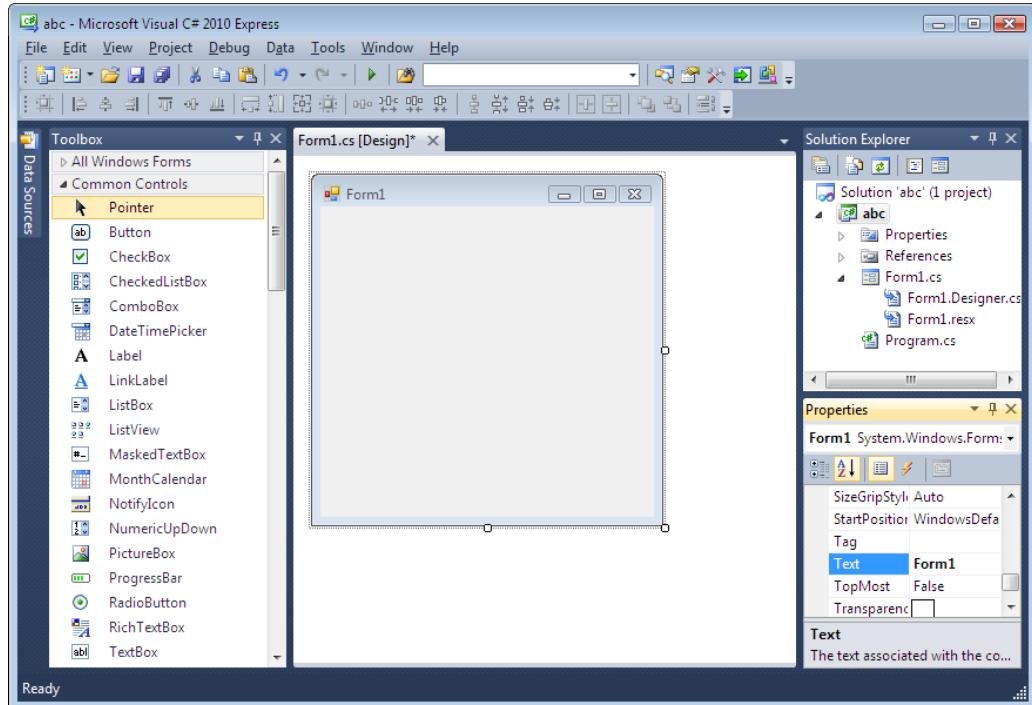


Рис. 1.2. Окно среды разработки Visual C# в начале работы над новым проектом Windows Forms



Рис. 1.3. Панель Standard

В левой части главного окна Visual C# отображается окно **Toolbox**. В нем (рис. 1.6) находятся *компоненты*. Компонент — это объект, реализующий некоторую функциональность. Например, в группе **Common Controls** находятся компоненты, реализующие пользовательский интерфейс (**Label** — область отображения текста; **TextBox** — поле редактирования текста; **Button** — командная кнопка), а в группе **Data** — компоненты, обеспечивающие доступ к базам данных.

В окне **Properties** (рис. 1.7) отображаются *свойства* выбранного в данный момент объекта — компонента или, если ни один из элементов (компонентов) формы не выбран, самой формы. Обратите внимание, в верхней части окна **Properties** отображаются имя выбранного объекта и его тип.

Окно **Properties** (свойства) используется для редактирования значений свойств объектов и, как следствие, изменения их вида и поведения. Например, результатом изменения значения свойства **Text** формы является изменение текста, находящегося в ее заголовке.

Свойства в окне **Properties** могут быть объединены в группы по функциональному признаку (**Categorized**) или упорядочены по алфавиту (**Alphabetical**). Чтобы изменить способ отображения, надо сделать щелчок на соответствующей кнопке, находящейся в верхней части окна **Properties** (рис. 1.8).

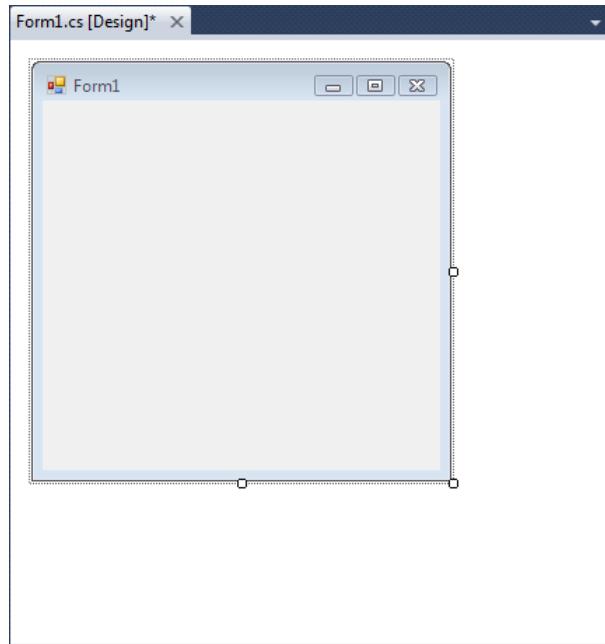


Рис. 1.4. Окно конструктора (дизайнера) формы

The screenshot shows the Microsoft Visual Studio Code Editor with three tabs open: "Form1.Designer.cs*", "Form1.cs*", and "Form1.cs [Design]*". The "Form1.cs [Design]" tab is active. The code editor displays the following C# code:

```
abc.Form1
    }
    base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do
/// the contents of this method with the code
/// </summary>
private void InitializeComponent()
{
    this.SuspendLayout();
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF();
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.FitClient;
    this.ClientSize = new System.Drawing.Size(292, 266);
    this.Name = "Form1";
    this.Text = "Form1";
    this.Load += new System.EventHandler(this.Form1_Load);
    this.ResumeLayout(false);
}

#endregion
```

Рис. 1.5. Инструкции, обеспечивающие создание формы, находятся в секции **Windows Form Designer generated code**

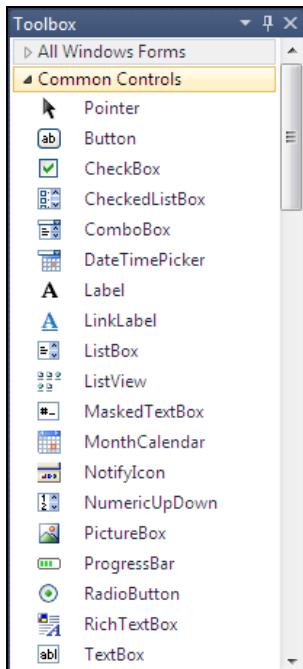


Рис. 1.6. Окно Toolbox

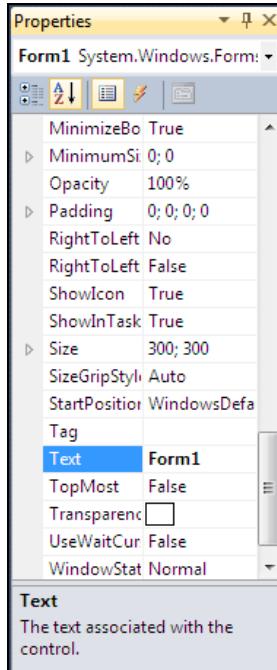


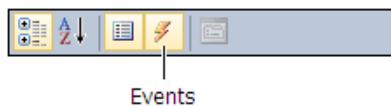
Рис. 1.7. Окно Properties



Categorized Alphabetical

Рис. 1.8. Кнопки управления способом группировки свойств

Сделав в окне **Properties** щелчок на кнопке **Events** (рис. 1.9), можно увидеть *события* (рис. 1.10), которые способен воспринимать выбранный объект (компонент или форма). Событие — это то, что происходит во время работы программы. Например, команная кнопка может реагировать на щелчок кнопкой мыши — событие *Click*.

Рис. 1.9. Чтобы увидеть события, которые способен воспринимать объект, нажмите кнопку **Events**

В окне **Solution Explorer** (рис. 1.11) отображается структура проекта (*solution* — решение, так часто называют приложение, обеспечивающее *решение* некоторой задачи). В нем перечислены файлы, образующие проект (подробно структура про-

екта рассматривается в главе 2). Окно **Solution Explorer** удобно использовать для быстрого доступа к нужному элементу проекта.

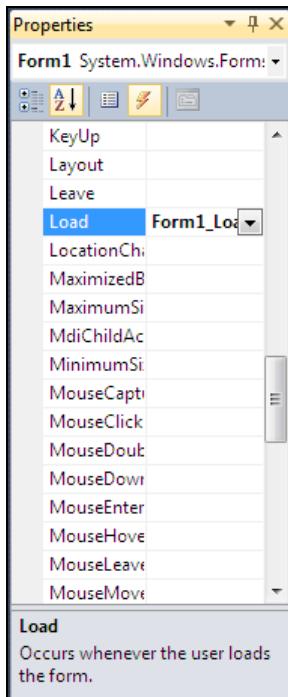


Рис. 1.10. События, которые может воспринимать объект (в данном случае — форма)

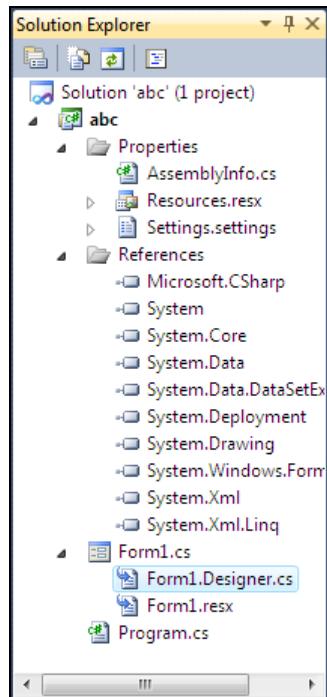


Рис. 1.11. В окне **Solution Explorer** отображается структура проекта

Справочная информация

Справочная информация Visual C# по умолчанию на компьютере программиста не устанавливается и, следовательно, недоступна. Чтобы получить доступ к справочной информации, надо выполнить настройку справочной системы.

Visual Studio позволяет программисту выбрать источник справочной информации — локальная справочная информация (находящаяся на компьютере разработчика) или online (предоставляемая Microsoft через Интернет).

Чтобы использовать локальную справочную информацию, надо:

1. В меню **Help** выбрать команду **Manage Help Settings**.
2. В окне **Help Library Manager** щелкнуть по ссылке **Choose online or local help** и в открывшемся окне выбрать **I want to use local help** (Использовать локальную справочную информацию).
3. В окне **Help Library Manager** щелкнуть по ссылке **Install Content from Online**, в появившемся окне выбрать (сделать щелчок на ссылке **Add**) раздел справочной информации, который надо установить, и нажать кнопку **Update** (рис. 1.12).

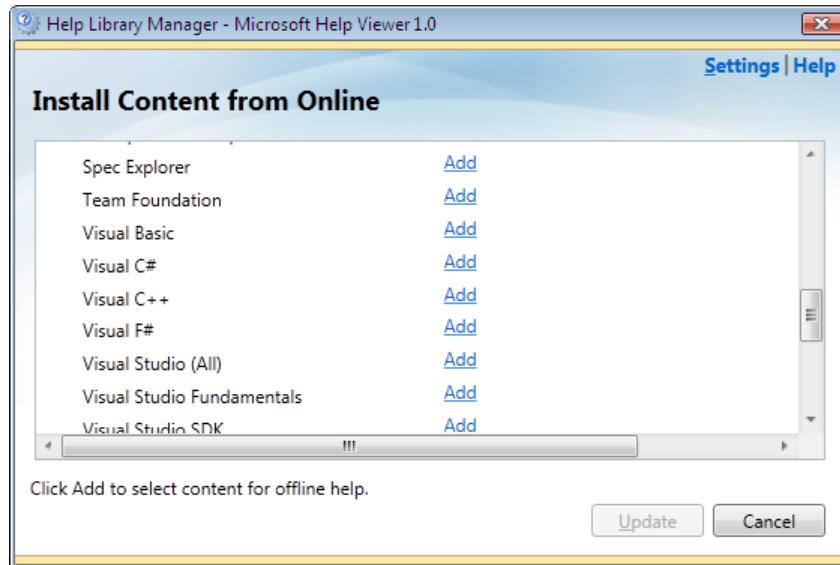
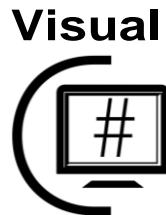


Рис. 1.12. Выбор раздела справочной информации, который надо установить на компьютер



ГЛАВА 2

Первый проект

Процесс разработки программы в Microsoft Visual C# рассмотрим на примере — создадим *приложение* (программы, предназначенные для решения прикладных задач, принято называть *приложениями*), позволяющее посчитать доход по вкладу (рис. 2.1).

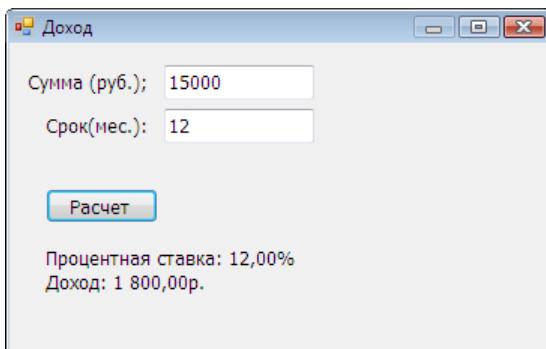


Рис. 2.1. Окно программы "Доход"

Начало работы над проектом

Чтобы начать работу над новым проектом, надо:

1. В меню **File** выбрать команду **New Project**.
2. В открывшемся окне **New Project** выбрать тип приложения — **Windows Forms Application - Visual C#**.
3. В поле **Name** ввести имя проекта — **profit** и нажать кнопку **OK** (рис. 2.2).

В результате описанных действий в папке временных проектов (по умолчанию это `C:\Users\User\AppData\Local\Temporary Projects`) будет создана папка `profit`, а в ней — проект `profit`.

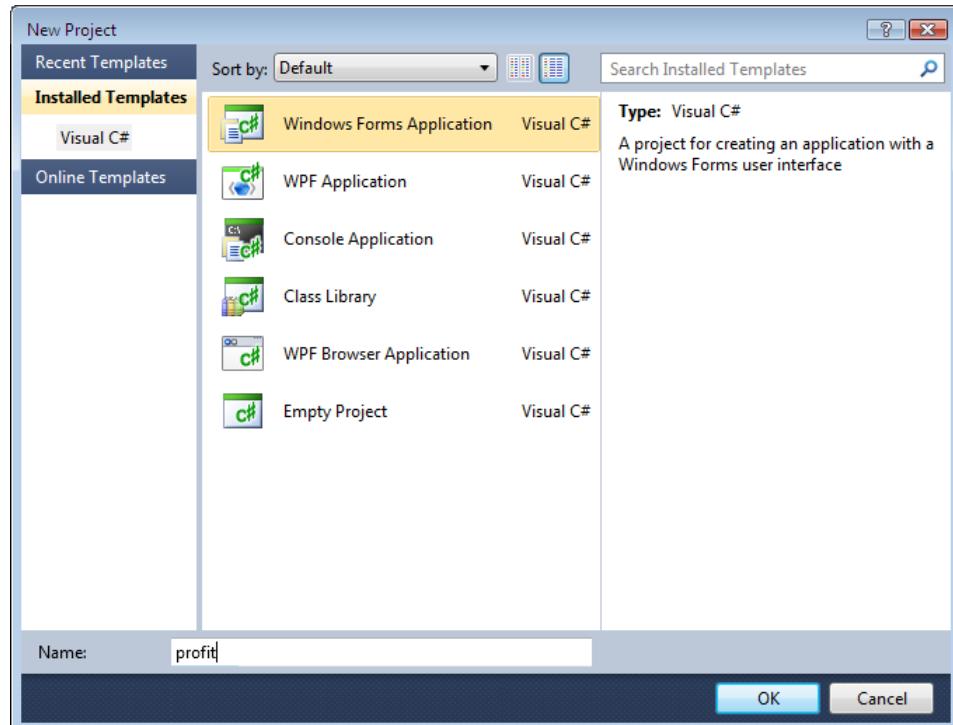


Рис. 2.2. Начало работы над новой программой

Форма

Работа над приложением начинается с создания стартовой *формы* — главного окна программы. Форма создается путем добавления на заготовку формы необходимых компонентов и изменения значений свойств самой формы.

Сначала нужно установить требуемые значения свойств формы, затем — поместить на форму необходимые *компоненты* (поля ввода информации, командные кнопки, поля отображения текста и др.) и выполнить.

Настройка формы (а также компонентов) осуществляется путем изменения значений *свойств*. Свойства *объекта* (формы, компонента) определяют его вид и поведение. Например, свойство *Text* определяет текст заголовка окна, а свойство *StartPosition* — положение окна в момент появления его на экране.

Основные свойства формы (объекта *Form*) приведены в табл. 2.1.

Таблица 2.1. Свойства формы (объекта *Form*)

Свойство	Описание
Name	Имя формы
Text	Текст в заголовке

Таблица 2.1 (окончание)

Свойство	Описание
Size	Размер формы. Уточняющее свойство <code>Width</code> определяет ширину, свойство <code>Height</code> — высоту
StartPosition	Положение формы в момент первого появления на экране. Форма может находиться в центре экрана (<code>CenterScreen</code>), в центре родительского окна (<code>CenterParent</code>). Если значение свойства равно <code>Manual</code> , то положение формы определяется значением свойства <code>Location</code>
Location	Положение формы на экране. Расстояние от верхней границы формы до верхней границы экрана задает уточняющее свойство <code>Y</code> , расстояние от левой границы формы до левой границы экрана — уточняющее свойство <code>X</code>
FormBorderStyle	Тип формы (границы). Форма может представлять собой обычное окно (<code>Sizable</code>), окно фиксированного размера (<code>FixedSingle</code> , <code>Fixed3D</code>), диалог (<code>FixedDialog</code>) или окно без кнопок Свернуть и Развернуть (<code>SizeableToolWindow</code> , <code>FixedToolWindow</code>). Если свойству присвоить значение <code>None</code> , у окна не будет заголовка и границы
ControlBox	Управляет отображением системного меню и кнопок управления окном. Если значение свойства равно <code>False</code> , то в заголовке окна кнопка системного меню, а также кнопки Свернуть , Развернуть , Закрыть не отображаются
MaximizeBox	Кнопка Развернуть . Если значение свойства равно <code>False</code> , то находящаяся в заголовке окна кнопка Развернуть недоступна
MinimizeBox	Кнопка Свернуть . Если значение свойства равно <code>False</code> , то находящаяся в заголовке окна кнопка Свернуть недоступна
Icon	Значок в заголовке окна
Font	Шрифт, используемый по умолчанию компонентами, находящимися на поверхности формы. Изменение значения свойства приводит к автоматическому изменению значения свойства <code>Font</code> всех компонентов формы (при условии, что значение свойства компонента не было задано явно)
ForeColor	Цвет, наследуемый компонентами формы и используемый ими для отображения текста. Изменение значения свойства приводит к автоматическому изменению соответствующего свойства всех компонентов формы (при условии, что значение свойства <code>Font</code> компонента не было задано явно)
BackColor	Цвет фона. Можно задать явно (выбрать на вкладке Custom или Web) или указать элемент цветовой схемы (выбрать на вкладке System)
Opacity	Степень прозрачности формы. Форма может быть непрозрачной (100%) или прозрачной. Если значение свойства меньше 100%, то сквозь форму видна поверхность, на которой она отображается

Для изменения значений свойств объектов используется окно **Properties**. В левой колонке окна перечислены свойства объекта, выбранного в данный момент, в правой — указаны значения свойств. Имя выбранного объекта отображается в верхней части окна **Properties**.

Чтобы в заголовке окна отображалось название программы, надо изменить значение свойства `Text`. Для этого следует щелкнуть левой кнопкой мыши в поле значение свойства `Text` (в поле появится курсор), ввести в поле редактирования текст **Доход** и нажать клавишу `<Enter>` (рис. 2.3).

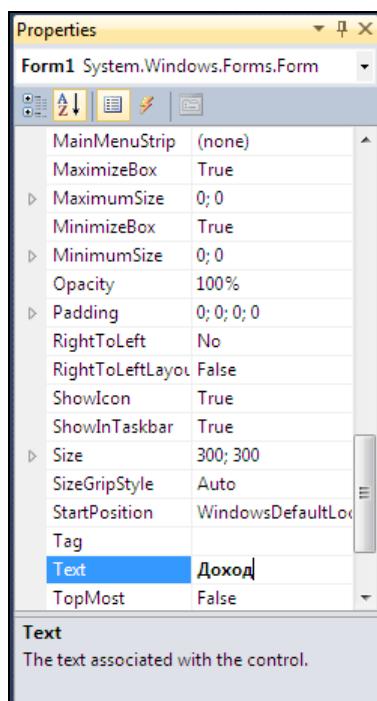


Рис. 2.3. Изменение значения свойства `Text` путем ввода нового значения

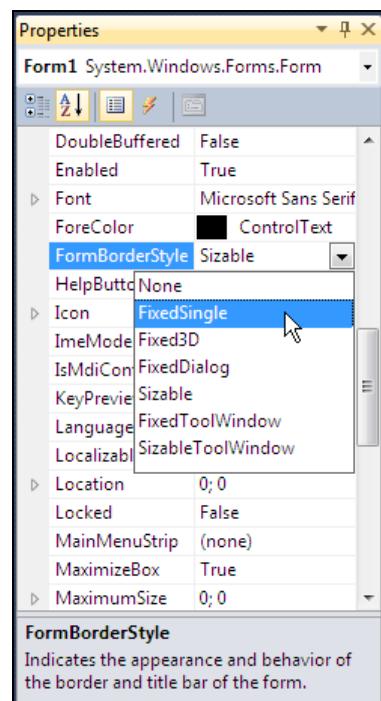


Рис. 2.4. Установка значения свойства путем выбора из списка

При выборе некоторых свойств, например `FormBorderStyle`, справа от текущего значения свойства появляется значок раскрывающегося списка. Очевидно, что значение таких свойств можно задать путем выбора из списка (рис. 2.4).

Некоторые свойства являются сложными. Они представляют собой совокупность других (уточняющих) свойств. Например, свойство `Size`, определяющее размер формы, представляет собой совокупность свойств `Width` и `Height`. Перед именами сложных свойств стоит значок ▶, в результате щелчка на котором раскрывается список уточняющих свойств (рис. 2.5). Значение уточняющего свойства можно задать (изменить) обычным образом — ввести нужное значение в поле редактирования.

Размер формы можно изменить и с помощью мыши, точно так же, как и любого окна, т. е. путем перемещения границы. По окончании перемещения границы значения свойств `Width` и `Height` будут соответствовать установленному размеру формы.

В результате выбора некоторых свойств, например `Font`, в поле значения свойства отображается кнопка, на которой изображены три точки. Это значит, что задать

значение свойства можно в дополнительном диалоговом окне, которое появится в результате щелчка на этой кнопке. Например, значение свойства **Font** можно задать путем ввода значений уточняющих свойств (**Name**, **Size**, **Style** и др.), а можно воспользоваться стандартным диалоговым окном **Шрифт**, которое появится в результате щелчка на кнопке с тремя точками (рис. 2.6).

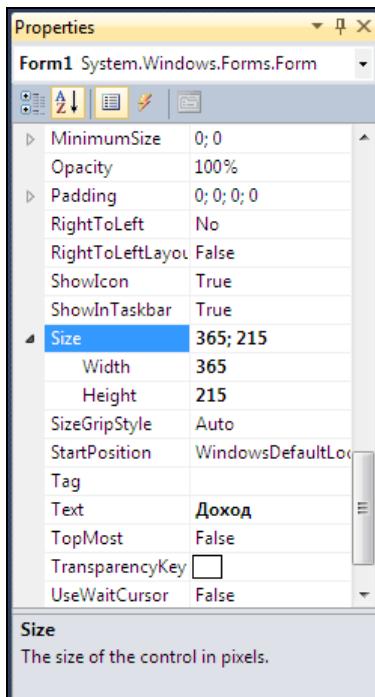


Рис. 2.5. Изменение значения уточняющего свойства

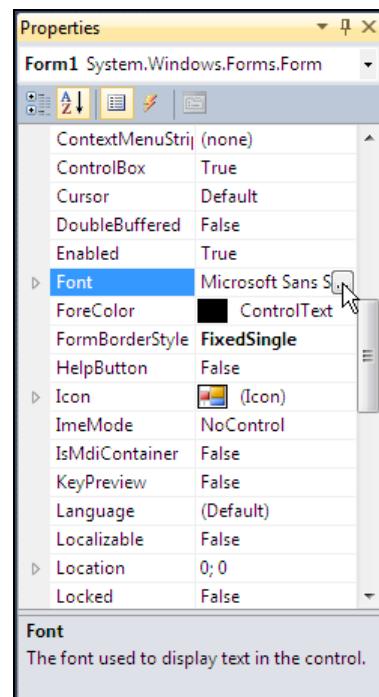


Рис. 2.6. Чтобы задать свойства шрифта, щелкните на кнопке с тремя точками

В табл. 2.2 приведены значения свойств формы программы "Доход". Значения остальных свойств формы оставлены без изменения и поэтому в таблице не представлены. Обратите внимание, в именах некоторых свойств есть точка. Это значит, что это значение уточняющего свойства.

Таблица 2.2. Значения свойств стартовой формы

Свойство	Значение	Комментарий
Text	Доход	
Size.Width	365	
Size.Height	215	
FormBorderStyle	FixedSingle	Тонкая граница формы. Во время работы программы пользователь не сможет изменить размер окна путем захвата и перемещения его границы

Таблица 2.2 (окончание)

Свойство	Значение	Комментарий
StartPosition	CenterScreen	Окно программы появится в центре экрана
MaximizeBox	False	В заголовке окна не отображать кнопку Развернуть
Font.Name	Tahoma	
Font.Size	10	

После того как будут установлены значения свойств формы, она должна выглядеть так, как показано на рис. 2.7. Теперь на форму надо добавить *компоненты*.

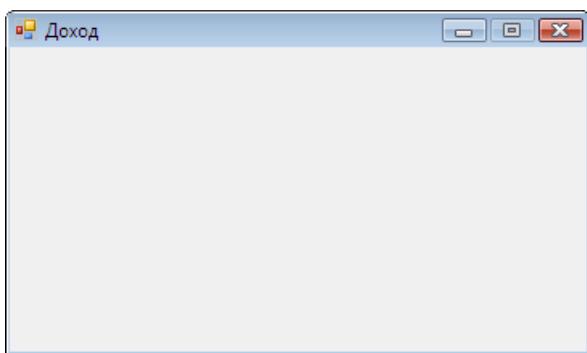


Рис. 2.7. Форма после изменения значений ее свойств

Компоненты

Поля ввода/редактирования, поля отображения текста, командные кнопки, списки, переключатели и другие элементы, обеспечивающие взаимодействие пользователя с программой, называют *компонентами пользовательского интерфейса*. Они находятся в окне **Toolbox** на вкладке **Common Controls**.

Программа "Доход" должна получить от пользователя исходные данные — сумму и срок вклада. Ввод данных с клавиатуры обеспечивает компонент **TextBox**. Таким образом, на форму разрабатываемого приложения нужно поместить два компонента **TextBox**.

Чтобы на форму добавить компонент **TextBox**, надо:

1. В палитре компонентов (окно **Toolbox**) раскрыть вкладку **Common Controls**.
2. Сделать щелчок на значке компонента **TextBox** (рис. 2.8).
3. Установить указатель мыши в ту точку формы, в которой должен быть левый верхний угол компонента, и сделать щелчок левой кнопкой мыши.

В результате на форме появляется поле ввода/редактирования — компонент **TextBox** (рис. 2.9).

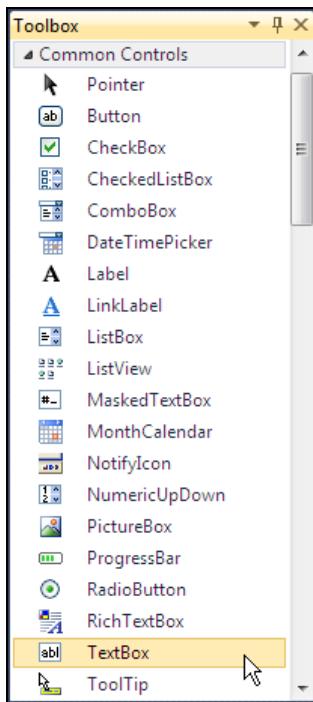


Рис. 2.8. Выбор компонента в палитре
(компонент TextBox — поле
редактирования)

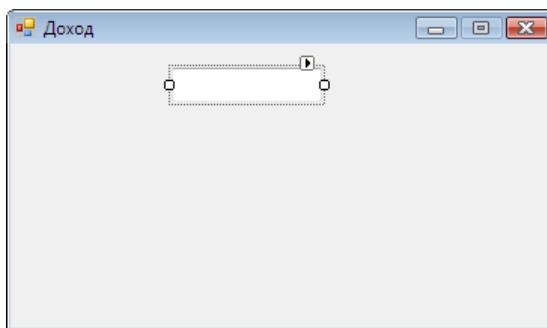


Рис. 2.9. Результат добавления
на форму компонента TextBox

Каждому добавленному компоненту среда разработки присваивает имя, которое состоит из названия компонента и его порядкового номера. Например, первый добавленный на форму компонент TextBox получает имя `textBox1`, второй — `textBox2`. Программист путем изменения значения свойства `Name` может поменять имя компонента. Однако в простых программах имена компонентов, как правило, не меняют.

Основные свойства компонента TextBox приведены в табл. 2.3.

Таблица 2.3. Свойства компонента TextBox

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
Text	Текст, который находится в поле редактирования
Location	Положение компонента на поверхности формы
Size	Размер компонента
Font	Шрифт, используемый для отображения текста в поле компонента
ForeColor	Цвет текста, находящегося в поле компонента
BackColor	Цвет фона поля компонента
BorderStyle	Вид рамки (границы) компонента. Граница компонента может быть обычной (Fixed3D), тонкой (FixedSingle) или отсутствовать (None)

Таблица 2.3 (окончание)

Свойство	Описание
TextAlign	Способ выравнивания текста в поле компонента. Текст в поле компонента может быть прижат к левой границе компонента (<code>Left</code>), правой (<code>Right</code>) или находится по центру (<code>Center</code>)
MaxLength	Максимальное количество символов, которое можно ввести в поле компонента
Multiline	Разрешает (<code>True</code>) или запрещает (<code>False</code>) ввод нескольких строк текста
ReadOnly	Разрешает (<code>True</code>) или запрещает (<code>False</code>) редактирование отображаемого текста
Lines	Массив строк, элементы которого содержат текст, находящийся в поле редактирования, если компонент находится в режиме <code>MultiLine</code> . Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
ScrollBars	Задает отображаемые полосы прокрутки: <code>Horizontal</code> — горизонтальная; <code>Vertical</code> — вертикальная; <code>Both</code> — горизонтальная и вертикальная; <code>None</code> — не отображать

На рис. 2.10 приведен вид формы программы "Доход" после добавления двух полей ввода/редактирования. Один из компонентов *выбран* — выделен рамкой. Свойства именно этого (выбранного) компонента отображаются в окне **Properties**. Чтобы увидеть и, если надо, изменить свойства другого компонента, нужно этот компонент выбрать — щелкнуть левой кнопкой мыши на изображении компонента в форме или выбрать его имя в раскрывающемся списке, который находится в верхней части окна **Properties** (рис. 2.11).

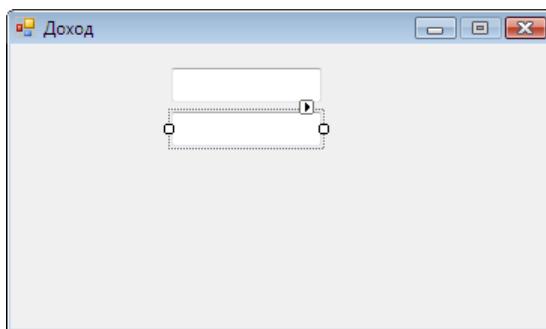


Рис. 2.10. Форма с двумя компонентами TextBox

Значения свойств компонента, определяющих размер и положение компонента на поверхности формы, можно изменить с помощью мыши.

Чтобы изменить положение компонента, необходимо установить курсор мыши на его изображение, нажать левую кнопку мыши и, удерживая ее нажатой, переместить компонент в нужную точку формы (рис. 2.12).

Для того чтобы изменить размер компонента, необходимо сделать щелчок на его изображении (в результате чего компонент будет выделен), установить указатель

мыши на один из маркеров, помечающих границу компонента, нажать левую кнопку мыши и, удерживая ее нажатой, изменить положение границы компонента (рис. 2.13).

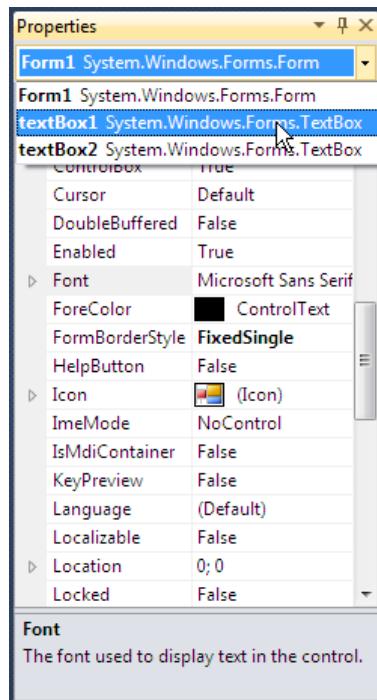


Рис. 2.11. Выбор компонента в окне Properties

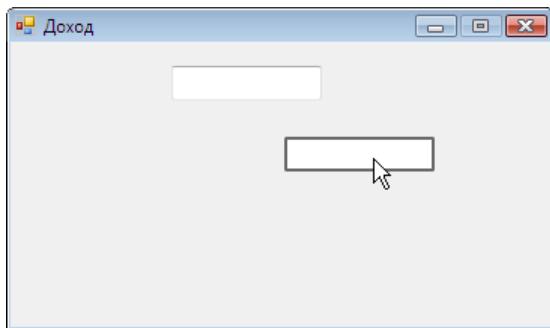


Рис. 2.12. Изменение положения компонента

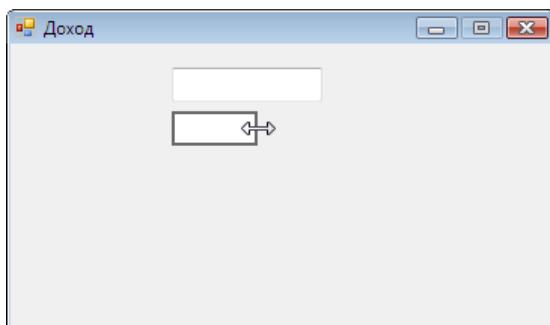


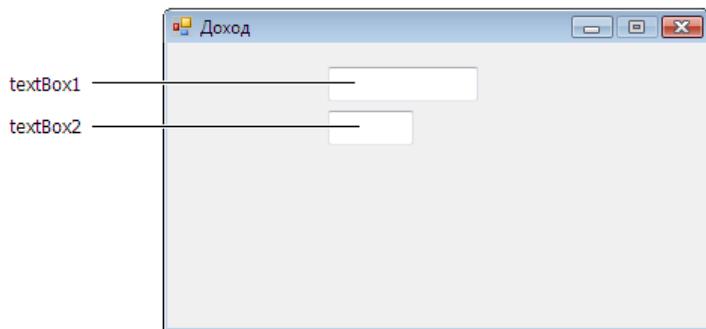
Рис. 2.13. Изменение размера компонента

В табл. 2.4 приведены значения свойств компонентов `textBox1` и `textBox2` (протирок показывает, что значением свойства `Text` является пустая строка). Значения остальных свойств компонентов оставлены без изменения и поэтому в таблице не показаны. Компонент `textBox1` предназначен для ввода суммы вклада, `textBox2` — срока. Так как значения свойства `Font` компонентов `TextBox` не были изменены, то во время работы программы текст в полях редактирования будет отображаться шрифтом, заданным для формы. Компоненты `TextBox`, как и другие компоненты, находящиеся на форме, наследуют значение свойства `Font` формы (если значение свойства `Font` компонента не было задано явно). Поэтому если изменить значение свойства `Font` формы, автоматически изменятся значения свойств `Font` компонентов, находящихся на форме. Если требуется, чтобы текст в поле компонента отображался другим шрифтом, нужно явно задать значение свойства `Font` этого компонента.

Форма программы "Доход" после настройки компонентов `TextBox` приведена на рис. 2.14.

Таблица 2.4. Значения свойств компонентов TextBox

Компонент	Свойство	Значение
textBox1	Location.X	107
	Location.Y	16
	Size.Width	100
	Size.Height	23
	Text	-
	TabIndex	0
textBox2	Location.X	107
	Location.Y	45
	Size.Width	57
	Size.Height	23
	Text	-
	TabIndex	1

**Рис. 2.14.** Форма после настройки компонентов TextBox

Отображение текста на поверхности формы (подсказок, результата расчета) обеспечивает компонент Label. В окне программы "Доход" текст отображается слева от полей ввода/редактирования (информация о назначении полей). Результат расчета также отображается в окне программы. Поэтому в форму надо добавить три компонента Label (рис. 2.15).

Добавляется компонент Label на форму точно так же, как и поле редактирования (компонент TextBox).

Основные свойства компонента Label приведены в табл. 2.5.

На форму разрабатываемого приложения надо добавить три компонента Label. В полях label1 и label2 отображается информация о назначении полей ввода, поле label3 используется для вывода результата расчета.

Значения свойств компонентов Label приведены в табл. 2.6.

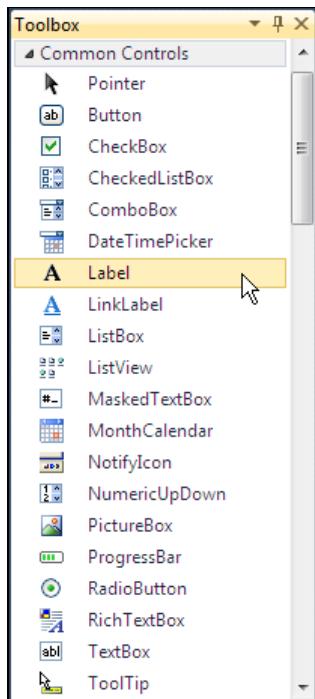


Рис. 2.15. Компонент Label — поле отображения текста

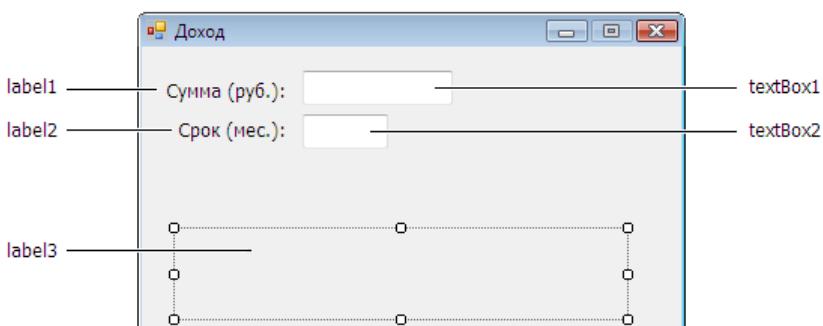
Таблица 2.5. Свойства компонента Label

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к свойствам компонента
Text	Отображаемый текст
Location	Положение компонента на поверхности формы
AutoSize	Признак автоматического изменения размера компонента. Если значение свойства равно <code>True</code> , то при изменении значения свойства <code>Text</code> (или <code>Font</code>) автоматически изменяется размер компонента
Size	Размер компонента (области отображения текста). Определяет (если значение свойства <code>AutoSize</code> равно <code>False</code>) размер компонента (области отображения текста)
Font	Шрифт, используемый для отображения текста
ForeColor	Цвет текста, отображаемого в поле компонента
BackColor	Цвет закраски области вывода текста
TextAlign	Способ выравнивания (расположения) текста в поле компонента. Всего существует девять способов расположения текста. На практике наиболее часто используют выравнивание по левой верхней границе (<code>TopLeft</code>), посередине (<code>TopCenter</code>) и по центру (<code>MiddleCenter</code>)

Таблица 2.6. Значения свойств компонентов Label

Компонент	Свойство	Значение
label1	Location.X	13
	Location.Y	19
	AutoSize	False
	Size.Width	88
	Size.Height	20
	Text	Сумма (руб.):
	TextAlign	MiddleRight
label2	Location.X	13
	Location.Y	45
	AutoSize	False
	Size.Width	88
	Size.Height	20
	Text	Срок (мес.):
	TextAlign	MiddleRight
label3	Location.X	23
	Location.Y	122
	AutoSize	False
	Size.Width	299
	Size.Height	50
	Text	-

После настройки компонентов Label форма разрабатываемого приложения должна выглядеть так, как показано на рис. 2.16.

**Рис. 2.16.** Вид формы после настройки полей отображения текста

Последнее, что надо сделать на этапе создания формы, — добавить на форму командную кнопку **Расчет**. Назначение этой кнопки очевидно.

Командная кнопка, компонент Button (рис. 2.17), добавляется на форму точно так же, как и другие компоненты. Значок компонента Button находится на вкладке **Common Controls**. Основные свойства компонента Button приведены в табл. 2.7.

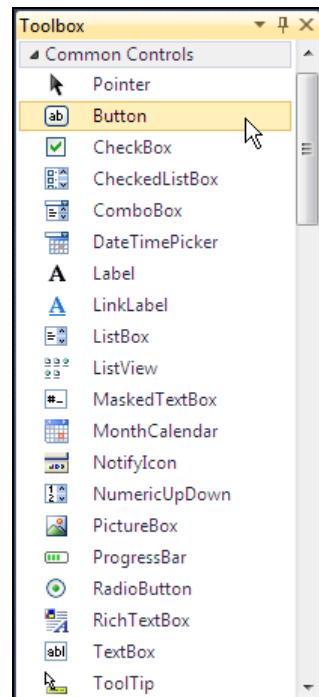


Рис. 2.17. Командная кнопка — компонент Button

Таблица 2.7. Свойства компонента Button

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
Text	Текст на кнопке
TextAlign	Положение текста на кнопке. Текст может располагаться в центре кнопки (MiddleCenter), быть прижат к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения надписи (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
FlatStyle	Стиль. Кнопка может быть стандартной (Standard), плоской (Flat) или "всплывающей" (Popup)
Location	Положение кнопки на поверхности формы. Уточняющее свойство X определяет расстояние от левой границы кнопки до левой границы формы, уточняющее свойство Y — от верхней границы кнопки до верхней границы клиентской области формы (нижней границы заголовка)
Size	Размер кнопки
Enabled	Признак доступности кнопки. Кнопка доступна, если значение свойства равно True, и недоступна, если значение свойства равно False (в этом случае нажать кнопку нельзя, событие Click в результате щелчка на ней не возникает)
Visible	Позволяет скрыть кнопку (False) или сделать ее видимой (True)
Cursor	Вид указателя мыши при позиционировании указателя на кнопке

Таблица 2.7 (окончание)

Свойство	Описание
Image	Картинка на поверхности кнопки. Рекомендуется использовать gif-файл, в котором определен прозрачный цвет
ImageAlign	Положение картинки на кнопке. Картинка может располагаться в центре (MiddleCenter), быть прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения картинки на кнопке (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
ImageList	Набор изображений, из которых может быть выбрано то, которое будет отображаться на поверхности кнопки. Представляет собой объект типа ImageList. Чтобы задать значение свойства, в форму приложения нужно добавить компонент ImageList
ImageIndex	Номер (индекс) изображения из набора ImageList, которое отображается на кнопке
ToolTip	Подсказка, появляющаяся рядом с указателем мыши при его позиционировании на кнопке. Чтобы свойство стало доступно, в форму приложения нужно добавить компонент ToolTip

После того как на форму будут добавлены кнопки, нужно выполнить их настройку. Значения свойств компонентов Button приведены в табл. 2.8, окончательный вид формы показан на рис. 2.18.

Таблица 2.8. Значения свойств компонента button1

Свойство	Значение
Location.X	26
Location.Y	84
Size.Width	75
Size.Height	23
Text	Расчет

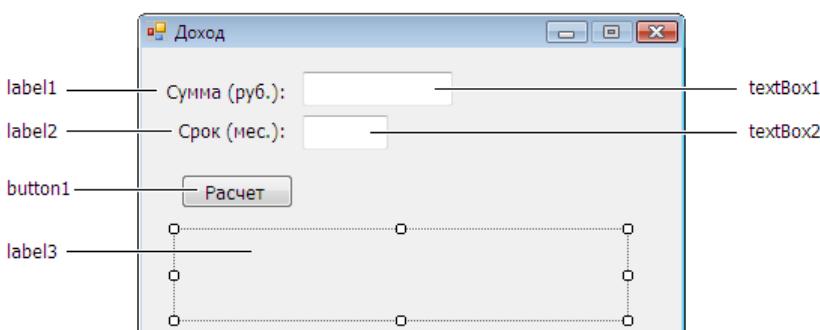


Рис. 2.18. Окончательный вид формы программы "Доход"

Завершив работу по созданию формы, можно приступить к программированию — созданию процедур обработки событий.

Событие

Вид формы программы "Доход" подсказывает, как работает программа. Очевидно, что пользователь должен ввести в поля редактирования исходные данные и сделать щелчок на кнопке **Расчет**. Щелчок на изображении командной кнопки — это пример того, что называется *событием*.

Событие (event) — это то, что происходит во время работы программы. Например, щелчок кнопкой мыши — это событие Click, двойной щелчок мышью — событие DblClick.

В табл. 2.9 приведены некоторые события, возникающие в результате действий пользователя.

Таблица 2.9. События

Событие	Описание
Click	Щелчок кнопкой мыши
DoubleClick	Двойной щелчок кнопкой мыши
MouseDown	Нажатие кнопки мыши
MouseUp	Отпускание нажатой кнопки мыши
MouseMove	Перемещение указателя мыши
KeyPress	Нажатие клавиши
KeyDown	Нажатие клавиши. События KeyDown и KeyPress — это чередующиеся, повторяющиеся события, которые происходят до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие KeyUp)
KeyUp	Отпускание нажатой клавиши
TextChanged	Признак, указывающий, изменился ли текст, находящийся в поле редактирования (изменилось значение свойства Text)
Load	Загрузка формы. Функция обработки этого события обычно используется для инициализации переменных, выполнения подготовительных действий
Paint	Событие происходит при появлении окна на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном
Enter	Получение фокуса элементом управления
Leave	Потеря фокуса элементом управления

Следует понимать, что одни и те же действия, но выполненные над разными объектами, вызывают разные события. Например, нажатие клавиши (событие KeyPress) в поле ввода/редактирования **Сумма** и нажатие клавиши (также событие KeyPress) в поле **Срок** — это два разных события.

Функция обработки события

Реакцией на событие должно быть какое-либо действие. В Visual C# реакция на событие реализуется как *функция обработки события*. Таким образом, для того чтобы программа в ответ на действия пользователя выполняла некоторую работу, программист должен написать функцию (метод) обработки соответствующего события.

Процесс создания функции обработки события рассмотрим на примере обработки события `Click` для кнопки **Расчет**.

Чтобы создать функцию обработки события, сначала надо выбрать компонент, для которого создается функция обработки события. Для этого в окне конструктора формы надо сделать щелчок левой кнопкой мыши на нужном компоненте. Затем в окне **Properties** щелчком на кнопке **Events** (рис. 2.19) нужно открыть вкладку **Events**.

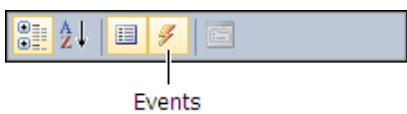


Рис. 2.19. Кнопка **Events**

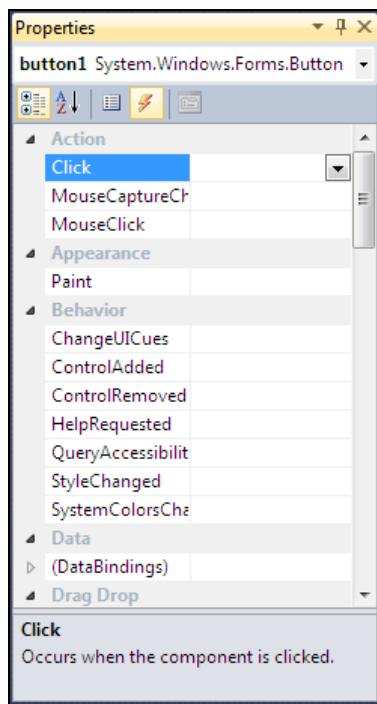


Рис. 2.20. На вкладке **Events** перечислены события, которые может воспринимать компонент

В левой колонке вкладки **Events** (рис. 2.20) перечислены события, которые может воспринимать выбранный компонент. Строго говоря, на вкладке **Events** указаны не события, а свойства, значением которых являются имена функций обработки соответствующих событий.

Для того чтобы создать функцию обработки события, нужно на вкладке **Events** выбрать событие (сделать щелчок мышью на имени события), в поле значения

свойства ввести имя функции обработки события (рис. 2.21) и нажать клавишу <Enter>.

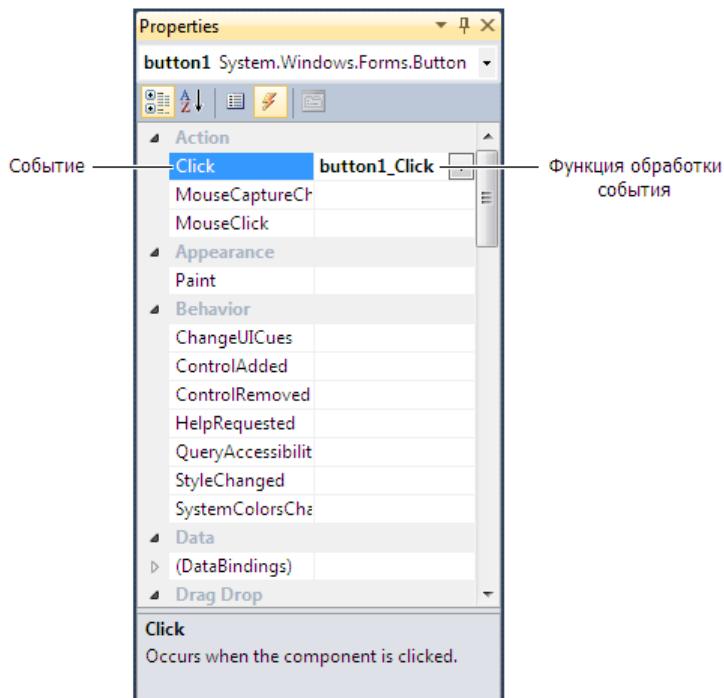


Рис. 2.21. Рядом с именем события надо ввести имя функции обработки события

В результате этих действий в модуль формы (cs-файл) будет добавлена функция (метод класса формы) обработки события и станет доступным окно редактора кода (рис. 2.22), в котором можно набирать инструкции, реализующие функцию обработки события.

Функция обработки события Click для кнопки **Расчет** (button1) приведена в листинге 2.1.

Листинг 2.1. Обработка события Click для кнопки **Расчет**

```
private void button1_Click(object sender, EventArgs e)
{
    double sum;      // сумма
    int period;     // срок

    double percent; // процентная ставка
    double profit;  // доход

    sum = System.Convert.ToDouble(textBox1.Text);
    period = System.Convert.ToInt32(textBox2.Text);
```

```
if (sum < 10000)
    percent = 8.5;
else
    percent = 12;

profit = sum * (percent/100/12) * period;

label3.Text = "Процентная ставка: " + percent.ToString("n") + "%\n" +
    "Доход: " + profit.ToString("c");
}
```

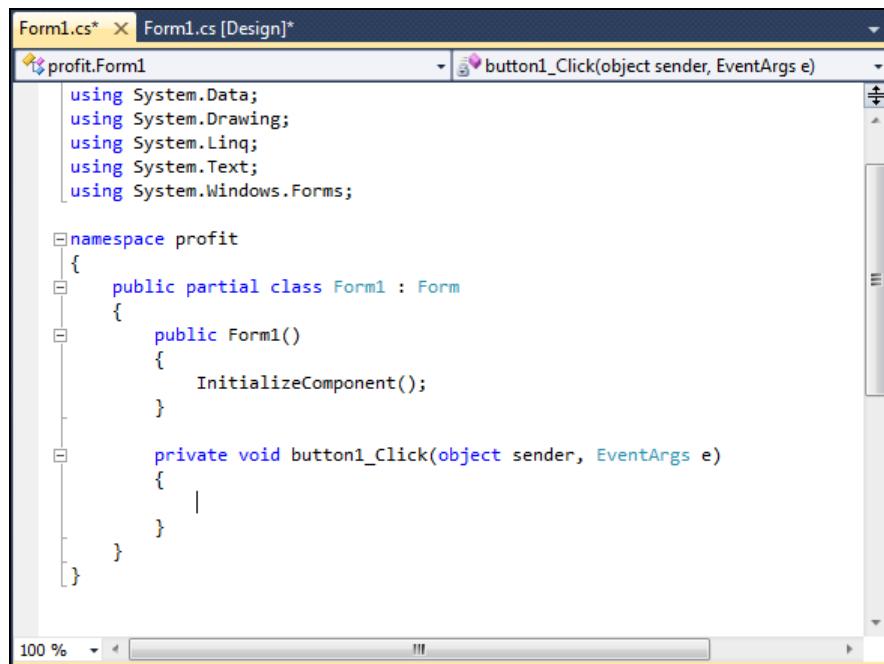


Рис. 2.22. Шаблон функции (метода) обработки события

Функция `button1_Click` вычисляет доход по вкладу и выводит результат расчета в поле компонента `label3`. Исходные данные (сумма и срок вклада) вводятся из полей редактирования `textBox1` и `textBox2` путем обращения к свойству `Text`. Значением свойства `Text` является строка, которая находится в поле редактирования. Свойство `Text` строкового типа, поэтому для преобразования строк в числа используются принадлежащие пространству имен `System.Convert` функции `ToDouble` и `ToInt32`. Следует обратить внимание, что функция `ToDouble` возвращает результат только в том случае, если строка, переданная ей в качестве параметра, является изображением дробного числа, что предполагает использование запятой в качестве десятичного разделителя (при стандартной для России настройке операционной системы). Аналогично, параметр функции `ToInt32` должен представлять собой строку, являющуюся изображением целого числа.

Другие функции преобразования строк приведены в табл. 2.10.

Таблица 2.10. Функции преобразования строк

Функция	Значение
ToSingle(s), ToDouble(s)	Дробное типа Single, Double
ToByte(s),ToInt16(s), ToInt32(s),ToInt64(s),	Целое типа Byte, Int16, Int32, Int64
ToUInt16(s),ToUInt32(s), ToInt64(s)	Целое типа UInt16, UInt32, UInt64

ПРОСТРАНСТВО ИМЕН

Концепция пространства имен является развитием концепции модулей. Пространство имен позволяет избежать конфликта имен, дает программисту свободу в выборе идентификаторов. Так, например, при объявлении функции можно не заботиться об уникальности ее имени, достаточно объявить эту функцию в новом пространстве имен.

В приведенной в листинге 2.1 функции для преобразования строки в дробное число используется функция `ToDouble`. Она принадлежит пространству имен `System.Convert`, на что указывает префикс перед именем функции (строго говоря, функция `ToDouble` — это метод объекта `Convert`, который принадлежит пространству имен `System`).

Пространство имен (namespace) — это контейнер (модуль), который предоставляет программе, использующей этот модуль, свои объекты (типы, функции, константы и т. д.). Например, пространство имен `System.Windows.Forms` содержит объекты `Label`, `TextBox`, `Button` и др.

Каждый объект является элементом какого-либо пространства имен. Например, поле редактирования, объект типа `TextBox`, является элементом или, как принято говорить, принадлежит пространству имен `System.Windows.Forms`.

Пространства имен, которые использует программа, указывается в инструкции `using`. Например, в начале модуля формы (cs-файл) есть ссылки на пространства имен `System`, `System.Windows.Forms`, `System.Drawing` и др.

Для того чтобы получить доступ к объекту пространства имен (например, методу или константе), следует перед именем объекта указать идентификатор пространства имен, которому принадлежит объект, разделив идентификатор и имя объекта точкой.

Например, инструкция

```
n = System.Convert.ToSingle(TextBox1.Text);
```

показывает, что для преобразования строки в число используется метод `ToSingle` объекта `Convert`, который принадлежит пространству имен `System`.

Вычисленные значения процентной ставки и величины дохода выводятся в поле `label3` путем присваивания значения свойству `Text`. Для преобразования дробного числа в строку (свойство `Text` строкового типа) используется функция (метод) `ToString`. Параметр метода `ToString` задает формат строки-результата: "`c`" — финансовый (от англ. *currency*); "`n`" — числовой (от англ. *number*). Следует обратить внимание, что при использовании финансового формата после числового значения

выводится обозначение денежной единицы (в соответствии с настройкой операционной системы). В табл. 2.11 приведены возможные форматы представления числовых информаций.

Таблица 2.11. Форматы представления чисел

Параметр функции <code>ToString</code>	Формат	Пример
" <code>C</code> "	<code>Currency</code> — финансовый (денежный). Используется для представления денежных величин. Обозначение денежной единицы, разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28 р.
" <code>e</code> "	<code>Scientific (exponential)</code> — научный. Используется для представления очень маленьких или очень больших чисел. Разделитель целой и дробной частей числа задается в настройках операционной системы	5,50528+E004
" <code>F</code> "	<code>Fixed</code> — число с фиксированным десятичным разделителем. Используется для представления дробных чисел. Количество цифр дробной части, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
" <code>N</code> "	<code>Number</code> — числовой. Используется для представления дробных чисел. Количество цифр дробной части, символ-разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
" <code>G</code> "	<code>General</code> — универсальный формат. Похож на <code>Number</code> , но разряды не разделены на группы	55055,275
" <code>R</code> "	<code>Roundtrip</code> — без округления. В отличие от формата <code>N</code> , этот формат не выполняет округления (количество цифр дробной части зависит от значения числа)	55 055,2775

Структура проекта

Проект представляет собой совокупность файлов, которые компилятор использует для создания выполняемого файла. Структура проекта отображается в окне **Solution Explorer** (рис. 2.23).

Основными элементами проекта являются:

- ◆ главный модуль приложения (файл `Program.css`);
- ◆ модули форм.

Главный модуль

В главном модуле находится функция `Main`, с которой начинается выполнение программы. Функция `Main` создает стартовую форму (имя класса стартовой формы

указывается в качестве параметра метода Run), в результате чего на экране появляется окно программы. Главный модуль программы "Доход" приведен в листинге 2.2.

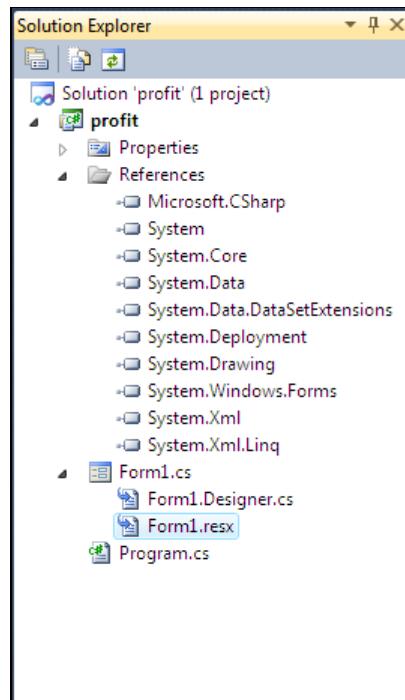


Рис. 2.23. Структура проекта отображается в окне **Solution Explorer**

Листинг 2.2. Главный модуль программы "Доход" (Program.cs)

```
using System.Windows.Forms;

namespace profit
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Модуль формы

Модуль формы содержит объявление класса формы. Физически модуль формы разделен на два файла: Form1.cs и Form1.Designer.cs (листинги 2.3 и 2.4 соответственно). В файле Form1.cs находятся функции (методы класса формы) обработки событий формы и ее компонентов. В файле Form1.Designer.cs (чтобы его увидеть, надо в окне **Solution Explorer** сделать двойной щелчок на имени файла) находится объявление класса формы, в том числе сформированная дизайнером формы функция InitializeComponent, обеспечивающая создание и настройку компонентов. Следует обратить внимание на секцию **Windows Form Designer generated code** (секция — фрагмент кода, находящийся между директивами #region и #endregion). В ней находится функция InitializeComponent, обеспечивающая непосредственно создание и инициализацию формы и компонентов.

Листинг 2.3. Модуль формы (файл Form1.cs)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace profit
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            double sum;           // сумма
            int period;          // срок

            double percent;      // процентная ставка
            double profit;       // доход

            sum = Convert.ToDouble(textBox1.Text);
            period = Convert.ToInt32(textBox2.Text);

            if (sum < 10000)
                percent = 8.5;
```

```
        else
            percent = 12;

        profit = sum * (percent/100/12) * period;

        label3.Text =
            "Процентная ставка: " + percent.ToString("n") + "%\n" +
            "Доход: " + profit.ToString("c");
    }
}
}
```

Листинг 2.4. Модуль формы (файл Form1.Designer.cs)

```
namespace profit
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be
        /// disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            ...
        }
    }
}
```

```
this.textBox1 = new System.Windows.Forms.TextBox();
this.textBox2 = new System.Windows.Forms.TextBox();
this.button1 = new System.Windows.Forms.Button();
this.label3 = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(9, 18);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(88, 16);
this.label1.TabIndex = 0;
this.label1.Text = "Сумма (руб.):";
//
// label2
//
this.label2.AutoSize = true;
this.label2.Location = new System.Drawing.Point(21, 47);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(76, 16);
this.label2.TabIndex = 1;
this.label2.Text = "Срок (мес.):";
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(103, 15);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(100, 23);
this.textBox1.TabIndex = 2;
//
// textBox2
//
this.textBox2.Location = new System.Drawing.Point(103, 44);
this.textBox2.Name = "textBox2";
this.textBox2.Size = new System.Drawing.Size(100, 23);
this.textBox2.TabIndex = 3;
//
// button1
//
this.button1.Location = new System.Drawing.Point(24, 97);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(75, 23);
this.button1.TabIndex = 4;
this.button1.Text = "Посчитать";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click +=
    new System.EventHandler(this.button1_Click);
```

```
//  
// label13  
//  
this.label3.Location = new System.Drawing.Point(21, 135);  
this.label3.Name = "label13";  
this.label3.Size = new System.Drawing.Size(228, 64);  
this.label3.TabIndex = 5;  
//  
// Form1  
//  
this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 16F);  
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;  
this.ClientSize = new System.Drawing.Size(359, 204);  
this.Controls.Add(this.label3);  
this.Controls.Add(this.button1);  
this.Controls.Add(this.textBox2);  
this.Controls.Add(this.textBox1);  
this.Controls.Add(this.label2);  
this.Controls.Add(this.label1);  
this.Font = new System.Drawing.Font  
    ("Tahoma", 9.75F, System.Drawing.FontStyle.Regular,  
     System.Drawing.GraphicsUnit.Point, ((byte)(204)));  
this.FormBorderStyle =  
    System.Windows.Forms.FormBorderStyle.FixedSingle;  
this.Margin = new System.Windows.Forms.Padding(3, 4, 3, 4);  
this.MinimizeBox = false;  
this.Name = "Form1";  
this.StartPosition =  
    System.Windows.Forms.FormStartPosition.CenterScreen;  
this.Text = "Доход";  
this.ResumeLayout(false);  
this.PerformLayout();  
  
}  
  
#endregion  
  
private System.Windows.Forms.Label label1;  
private System.Windows.Forms.Label label2;  
private System.Windows.Forms.TextBox textBox1;  
private System.Windows.Forms.TextBox textBox2;  
private System.Windows.Forms.Button button1;  
private System.Windows.Forms.Label label3;  
}  
}
```

Сохранение проекта

Как было сказано раньше, в момент создания проекта среда разработки в папке временных проектов (по умолчанию это C:\Users\User\AppData\Local\Temporary Projects, где **User** — имя пользователя в системе) создает каталог проекта. Чтобы программист мог работать с программой в дальнейшем, проект надо сохранить явно. Для этого в меню **File** надо выбрать команду **Save All** и в появившемся окне **Save Project** нажать кнопку **Save** (рис. 2.24).

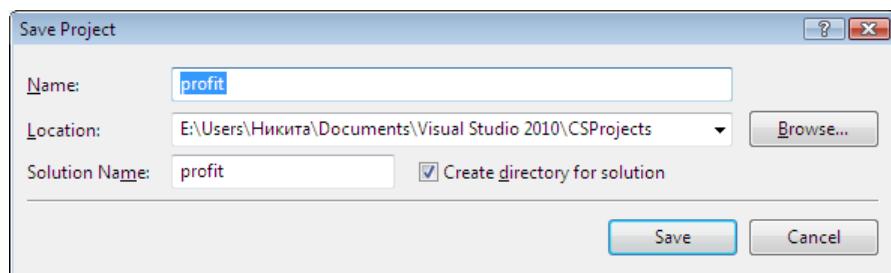


Рис. 2.24. Сохранение проекта

Следует обратить внимание, что в момент сохранения проекта в папке, имя которой указано в поле **Location**, для сохраняемого проекта будет создана новая папка (если установлен флажок **Create directory for solution**).

Компиляция

Процесс преобразования исходной программы в выполняемую называется *компиляцией* или построением (build). Укрупненно процесс построения программы можно представить как последовательность двух этапов: компиляция и компоновка. На этапе компиляции выполняется перевод исходной программы (модулей) в некоторое внутреннее представление. На этапе компоновки — объединение модулей в единую программу.

Процесс построения программы активизируется в результате выбора в меню **Debug** команды **Build solution**, а также в результате запуска программы из среды разработки (меню **Debug**, команда **Start Debugging**), если с момента последней компиляции в программу были внесены изменения.

Результат компиляции отражается в окне **Error List**. Если в программе нет ошибок, то по завершении процесса компиляции окно **Error List** выглядит так, как показано на рис. 2.25.

Если в процессе построения в программе обнаруживаются ошибки, то в окне **Error List** (рис. 2.26) выводится их список. Чтобы перейти к фрагменту кода, содержащего ошибку, надо сделать двойной щелчок левой кнопкой мыши в строке сообщения об этой ошибке.

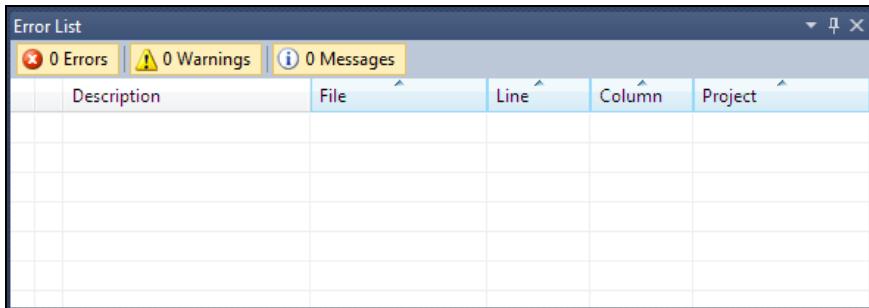


Рис. 2.25. Результат построения (в программе ошибок нет)

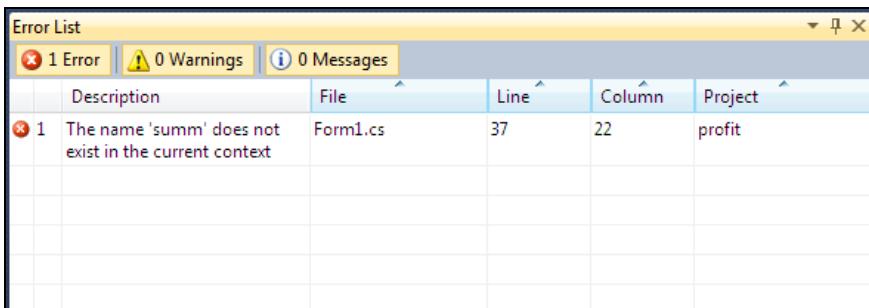


Рис. 2.26. Результат построения (в программе есть ошибка)

Ошибки

Компилятор генерирует выполняемую программу (exe-файл) только в том случае, если в исходной программе (в тексте) нет ошибок.

Если в программе есть ошибки, то программист должен их устранить. Процесс устранения ошибок носит итерационный характер. Обычно сначала устраняются наиболее очевидные ошибки, например, объявляются необъявленные переменные, затем, после выполнения повторной компиляции, — остальные.

В табл. 2.12 приведены сообщения компилятора о типичных ошибках.

Таблица 2.12. Сообщения компилятора об ошибках

Сообщение компилятора	Вероятная причина ошибки
The name <i>идентификатор</i> does not exist in the current context (В текущем контексте имя не существует)	1. Используемая в программе переменная не объявлена. 2. Ошибка при записи имени переменной. Например, объявлена переменная <i>sum</i> , а в тексте программы написано: <i>Sum</i>
Cannot implicitly convert type <i>type1</i> to <i>type2</i> (Невозможно преобразовать значение типа <i>type1</i> в значение типа <i>type2</i>) Пример: Cannot implicitly convert type 'double' to 'int'	В инструкции присваивания тип выражения не соответствует типу переменной, которой присваивается значение. Например, если переменные <i>n</i> и <i>m</i> целого типа, то инструкция <i>n = m/12</i> неверная, т. к. выражение <i>m/12</i> дробное

Таблица 2.12 (окончание)

Сообщение компилятора	Вероятная причина ошибки
Use of unassigned local variable (Используется локальная переменная, которой не присвоено начальное значение)	В программе нет инструкции, присваивающей переменной начальное значение
Пространство_имен does not contain definition for Идентификатор (Пространство имен Пространство_имен не содержит определение идентификатора Идентификатор)	Неправильно (например, не в том регистре) записано имя пространства имен или идентификатор (например, имя функции) ему принадлежащий. Пример: <code>System.Convert.ToInt32(textBox1.Text)</code> — неправильно записано имя функции. Должно быть <code>ToInt32</code>
'; ' expected (ожидается символ "точка с запятой")	После инструкции нет символа "точка с запятой"

Следует обратить внимание на то, что компилятор языка C# *различает* прописные и строчные буквы. Запись имени переменной или функции "не в том регистре" — типичная причина ошибок в программе.

Предупреждения

В программе могут быть не только ошибки, но и неточности. Например, инструкция присваивания целой переменной дробного значения формально является верной. Присвоить значение переменной можно, но что делать с дробной частью? Отбросить или округлить? Что хотел сделать программист, записав эту инструкцию?

При обнаружении в программе неточностей компилятор выводит предупреждения — Warnings. Например, при обнаружении не объявленной, но не используемой переменной выводится сообщение: `unreferenced local variable`. Действительно, зачем объявлять переменную и не использовать ее?

В табл. 2.13 приведены предупреждения и подсказки компилятора о типичных неточностях в программе.

Таблица 2.13. Предупреждения и подсказки компилятора

Сообщение	Причина
The variable ... is declared but never used	Переменная объявлена, но не используется
The variable ... is assigned but its value is never used	Переменной присвоено значение, но оно не используется

Запуск программы

Пробный запуск программы можно выполнить из Visual Studio, не завершая работу со средой разработки. Для этого в меню **Debug** надо выбрать команду **Start Debugging**. Можно также сделать щелчок на находящейся в панели инструментов **Debug** кнопке **Start Debugging** (рис. 2.27) или нажать клавишу <F5>.



Рис. 2.27. Чтобы запустить программу, сделайте щелчок на кнопке **Start Debugging**

Исключения

Ошибки, возникающие во время работы программы, называют *исключениями*. В большинстве случаев причиной исключений (exception) являются неверные данные. Например, если в поле **Сумма** окна программы "Доход" ввести, скажем, 100.50 и сделать щелчок на кнопке **Расчет**, то на экране появится окно (рис. 2.28) с сообщением о возникновении исключения `FormatException`: "Input string was not in correct format" ("Неверный формат введенной строки"). Кроме этого, среда разработки делает активным окно редактора кода, в котором выделяется инструкция программы, при выполнении которой произошла ошибка (возникло исключение).

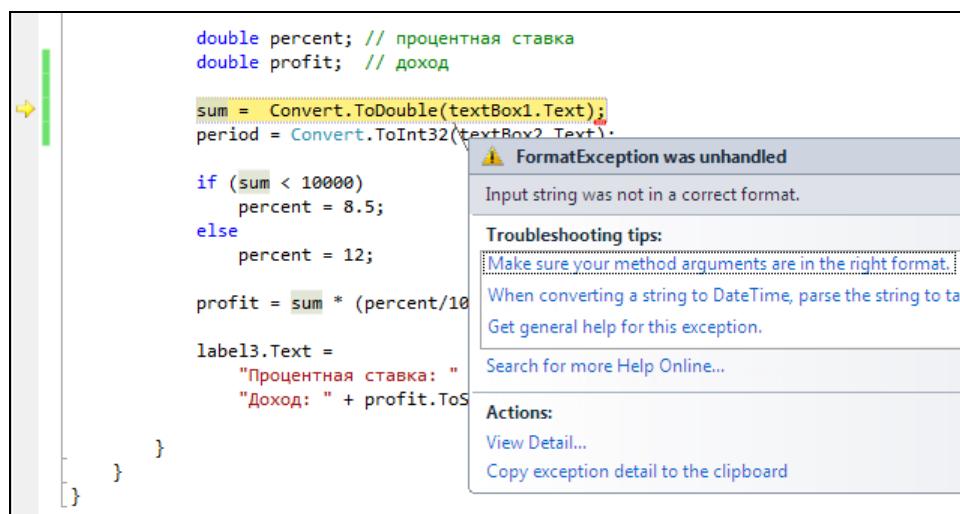


Рис. 2.28. Пример сообщения об исключении — ошибке, произошедшей во время работы программы (программа запущена из среды разработки)

Причина возникновения исключения в рассматриваемом примере в следующем. Преобразование строки в число выполняет функция `ToDouble`. Эта функция работа-

ет правильно, если ее параметром действительно является строковое представление дробного числа, что при стандартной для России настройке операционной системы предполагает использование в качестве десятичного разделителя *запятой*. В рассматриваемом примере строка 100.50 не является строковым представлением дробного числа, т. к. в качестве десятичного разделителя указана *точка*, и, поэтому, возникает исключение `FormatException` — ошибка формата. Исключение "ошибка формата" произойдет и в том случае, если в поле **Срок** будет введено дробное значение. Причина — попытка преобразовать в целое значение строку, которая не является изображением целого числа. Это же исключение произойдет и в том случае, если какое-либо из полей ввода оставить незаполненным.

Для того чтобы остановить программу, во время работы которой возникло исключение, надо в меню **Debug** выбрать команду **Stop Debugging** или нажать комбинацию клавиш `<Shift>+<F5>`.

Если программа запущена из операционной системы (из Windows), то при возникновении исключения так же, как и в случае запуска программы из среды разработки, выводится сообщение об ошибке (рис. 2.29). Чтобы остановить работу программы, надо нажать кнопку **Quit**. Щелчок на кнопке **Continue** разрешает продолжить выполнение программы, несмотря на возникшую ошибку.

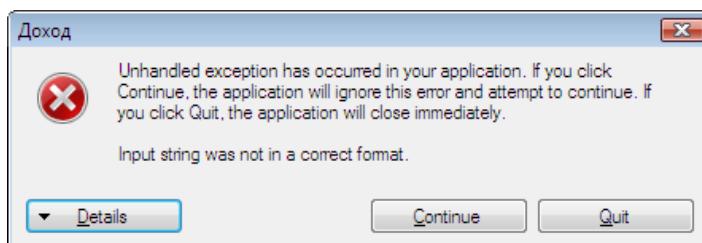


Рис. 2.29. Пример сообщения о возникновении исключения (программа запущена из операционной системы)

Обработка исключений

По умолчанию обработку исключений берет на себя автоматически добавляемый в выполняемую программу код, который обеспечивает вывод сообщения об ошибке и завершение работы программы, при выполнении которой возникло исключение. Вместе с тем программист может поместить в программу код, который выполнит обработку исключения.

В простейшем случае инструкция обработки исключения в общем виде выглядит так:

```
try
{
    // Здесь инструкции, при выполнении которых
    // может возникнуть исключение
}
```

```
catch (TypeException e)
{
    // Здесь инструкции обработки исключения
}
```

Ключевое слово `try` указывает, что далее следуют инструкции, при выполнении которых возможно возникновение исключений, и что обработку этих исключений берет на себя программа. Слово `catch` обозначает начало секции обработки исключений. После слова `catch` указывается тип исключения, обработку которого берет на себя программа. Далее следуют инструкции, обеспечивающие обработку исключения. Нужно обратить внимание на то, что инструкции секции `try`, следующие за той, при выполнении которой возникло исключение, после обработки исключения не выполняются.

В табл. 2.14 перечислены некоторые из возможных исключений и указаны вероятные причины их возникновения.

Таблица 2.14. Типичные исключения

Исключение	Возникает
<code>FormatException</code> — ошибка формата (преобразования)	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому типу. Наиболее часто возникает при преобразовании строки символов в число
<code>IndexOutOfRangeException</code> — выход значения индекса за допустимые границы	При обращении к несуществующему элементу массива
<code>ArgumentOutOfRangeException</code> — выход значения аргумента за допустимые границы	При обращении к несуществующему элементу данных, например, при выполнении операций со строками
<code>OverflowException</code> — переполнение	Если результат выполнения операции выходит за границы допустимого диапазона, а также при выполнении операции деления, если делитель равен нулю

В качестве примера обработки исключения в листинге 2.5 приведена функция обработки события `Click` для кнопки **Расчет** программы "Доход". При возникновении исключения `FormatException` программа определяет причину (какое из полей формы незаполнено или содержит неверные данные) и выводит соответствующее сообщение (рис. 2.30).

Листинг 2.5. Щелчок на кнопке **Расчет** (пример обработки исключения)

```
private void button1_Click(object sender, EventArgs e)
{
    double sum;        // сумма
    int period;        // срок

    double percent;   // процентная ставка
    double profit;    // доход
```

```

try
{
    sum = Convert.ToDouble(textBox1.Text);
    period = Convert.ToInt32(textBox2.Text);
    if (sum < 10000)
        percent = 8.5;
    else
        percent = 12;

    profit = sum * (percent / 100 / 12) * period;

    label3.Text =
        "Процентная ставка: " + percent.ToString("n") + "%\n" +
        "Доход: " + profit.ToString("c");
}

catch (FormatException ex)
{
    // MessageBox.Show(ex.Message);
    if ((textBox1.Text.Length == 0) || (textBox1.Text.Length == 0))
        MessageBox.Show("Оба поля должны быть заполнены.",
                        "Доход",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
    else
        MessageBox.Show("Ошибка в исходных данных. "+
                        "В поле Сумма надо ввести целое или дробное число " +
                        "(в качестве десятичного разделителя используйте " +
                        "запятую), в поле Срок - целое.",
                        "Доход",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
}
}

```

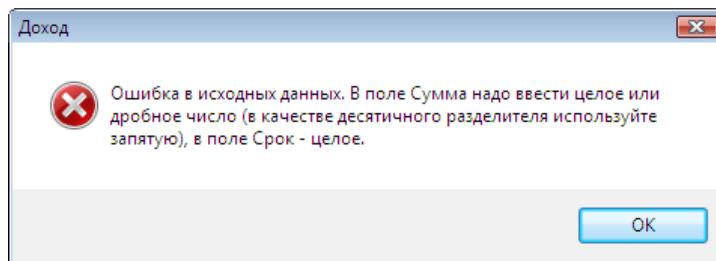


Рис. 2.30. Пример сообщения об ошибке

В приведенной функции обработки события click для вывода сообщения о неверных данных используется функция MessageBox.Show, инструкция вызова которой в общем виде выглядит так:

```
r = MessageBox.Show(Сообщение, Заголовок, Кнопки, ТипСообщения,
                    КнопкаПоУмолчанию)
```

где:

- ◆ *Сообщение* — текст сообщения;
- ◆ *Заголовок* — текст в заголовке окна сообщения;
- ◆ *Кнопки* — кнопки, отображаемые в окне сообщения (табл. 2.15);
- ◆ *Тип* — тип сообщения. Сообщение может быть информационным, предупреждающим или сообщением об ошибке. Каждому типу сообщения соответствует значок (табл. 2.16);
- ◆ *КнопкаПоУмолчанию* — порядковый номер кнопки, на которой находится фокус при появлении окна сообщения на экране (табл. 2.17).

Значение (тип `System.Windows.Forms.DialogResult`), возвращаемое функцией `MessageBox.Show`, позволяет определить, какая кнопка была нажата пользователем для завершения диалога (табл. 2.18). Если в окне сообщения отображается одна кнопка (очевидно, что в этом случае не нужно проверять, какую кнопку нажал пользователь), то функцию `MessageBox.Show` можно вызвать как процедуру.

Таблица 2.15. Идентификаторы кнопок

Значение параметра	Кнопки, отображаемые в окне сообщения
<code>MessageBoxButtons.OK</code>	OK
<code>MessageBoxButtons.YesNo</code>	Yes, No (Да, Нет)
<code>MessageBoxButtons.YesNoCancel</code>	Yes, No, Cancel (Да, Нет, Отменить)

Таблица 2.16. Тип сообщения

Сообщение	Тип сообщения	Значок
Warning — Внимание	<code>MessageBoxIcon.Warning</code>	
Error — Ошибка	<code>MessageBoxIcon.Error</code>	
Information — Информация	<code>MessageBoxIcon.Information</code>	

Таблица 2.17. Активная кнопка

Значение параметра	Номер активной кнопки
<code>System.Windows.Forms.MessageBoxDefaultButton.Button1</code>	1
<code>System.Windows.Forms.MessageBoxDefaultButton.Button2</code>	2
<code>System.Windows.Forms.MessageBoxDefaultButton.Button3</code>	3

Таблица 2.18. Значения функции `MessageBox.Show`

Значение	Нажата кнопка
<code>System.Windows.Forms.DialogResult.Yes</code>	Yes
<code>System.Windows.Forms.DialogResult.No</code>	No
<code>System.Windows.Forms.DialogResult.Cancel</code>	Cancel

Внесение изменений

Программу "Доход" можно усовершенствовать. Например, сделать так, чтобы в поля редактирования пользователь мог ввести только числа (в поле **Сумма** — дробное число, в поле **Срок** — целое), чтобы в результате нажатия клавиши <Enter> в поле **Сумма** курсор переходил в поле **Срок**, а при нажатии этой же клавиши в поле **Срок** становилась активной кнопка **Расчет**. Кроме этого, можно сделать так, чтобы кнопка **Расчет** становилась доступной только после ввода данных в оба поля редактирования.

Чтобы внести изменения в программу, нужно открыть соответствующий проект. Для этого надо в меню **File** выбрать команду **Open Project**, открыть папку проекта и сделать щелчок на значке файла проекта (рис. 2.31).

Нужный проект для загрузки можно выбрать также из списка проектов, над которыми в последнее время работал программист. Этот список становится доступным в результате выбора в меню **File** команды **Recent Projects and Solutions**.

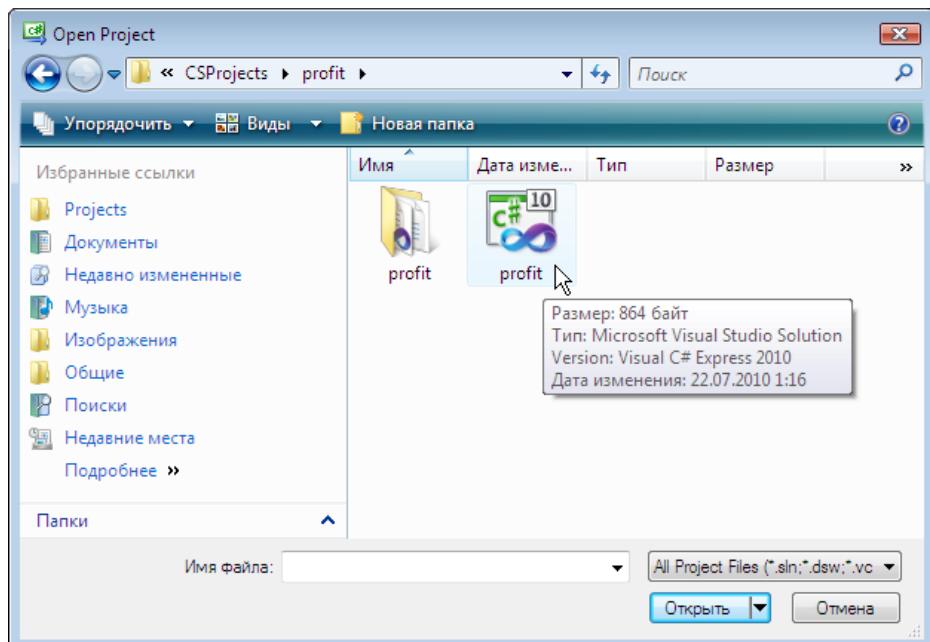


Рис. 2.31. Загрузка проекта

Чтобы программа "Доход" работала так, как было описано ранее, надо создать функции обработки событий `KeyPress` и `TextChanged` для полей редактирования (компонентов `textBox1` и `textBox2`). Функция обработки события `KeyPress` (для каждого компонента свой) фильтрует символы, вводимые пользователем. Она проверяет символ нажатой клавиши (символ передается в процедуру обработки события через параметр `e`) и, если символ "запрещен", присваивает значение `True` свойству `Handled`. В результате "запрещенный" символ в поле редактирования не появляется. Процедура обработки события `TextChanged` (событие возникает, если текст, находящийся в поле редактирования, изменился, например, в результате нажатия какой-либо клавиши в поле редактирования) управляет доступностью кнопки **Расчет**. Она проверяет, есть ли данные в полях редактирования, и, если в каком-либо из полей данных нет, присваивает свойству `Enabled` кнопки `button1` значение `False` (тем самым делает кнопку недоступной). Следует обратить внимание, что действие, которое надо выполнить, если изменилось содержимое поля `textBox1`, ничем не отличается от действия, которое надо выполнить, если изменилось содержимое поля `textBox2`. Поэтому обработку события `TextChanged` для обоих компонентов может выполнить *одна* функция. Чтобы *одна* функция могла обрабатывать события *разных* компонентов, сначала надо создать функцию обработки события для одного компонента, а затем указать эту функцию в качестве обработчика соответствующего события другого компонента (рис. 2.32).

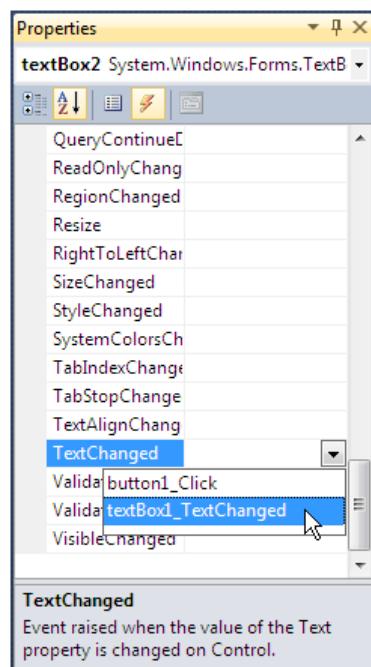


Рис. 2.32. Выбор функции обработки события: обработку события `TextChanged` компонента `textBox2` выполняет функция обработки этого же события компонента `textBox1`

Функции обработки указанных событий приведены в листинге 2.6. Обратите внимание, теперь в функции обработки события `Click` кнопки **Расчет** нет инструкций обработки исключений. Теперь они не нужны. Исключения не могут возникнуть, потому что пользователь не сможет ввести в поля редактирования неверные данные, а кнопка **Расчет** становится доступной только после того, как будут заполнены оба поля.

Листинг 2.6. Функции обработки событий

```
// щелчок на кнопке Расчет
private void button1_Click(object sender, EventArgs e)
{
    double sum;           // сумма
    int period;         // срок

    double percent; // процентная ставка
    double profit;  // доход

    sum = Convert.ToDouble(textBox1.Text);
    period = Convert.ToInt32(textBox2.Text);
    if (sum < 10000)
        percent = 8.5;
    else
        percent = 12;

    profit = sum * (percent / 100 / 12) * period;

    label3.Text = "Процентная ставка: " + percent.ToString("n") + "%\n" +
                  "Доход: " + profit.ToString("c");
}

// нажатие клавиши в поле Сумма
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9')) // цифра
        return;

    // "Правильный" десятичный разделитель – запятая.
    // Заменим точку запятой
    if (e.KeyChar == '.') e.KeyChar = ',';

    if (e.KeyChar == ',')
    {
        // Нажата запятая. Проверим,
        // может, запятая уже есть в поле редактирования
        if ((textBox1.Text.IndexOf(',') != -1) ||
              (textBox1.Text.Length == 0))
        {
            // Запятая уже есть.
            // Запретить ввод еще одной
            e.Handled = true;
        }
    }
    return;
}
```

```
if (Char.IsControl (e.KeyChar))
{
    if (e.KeyChar == (char)Keys.Enter)
    {
        // Нажата клавиша <Enter>.
        // Переместить курсор в поле Срок
        textBox2.Focus ();
    }
    return;
}

// остальные символы запрещены
e.Handled = true;
}

// нажатие клавиш в поле Срок
private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
{
    // в поле Срок можно ввести только целое число
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
        // цифра
        return;

    if (Char.IsControl (e.KeyChar))
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            // Нажата клавиша <Enter>.
            // Переместить фокус на кнопку Расчет
            button1.Focus ();
        }
        return;
    }

    // остальные символы запрещены
    e.Handled = true;
}

// Эта функция обрабатывает событие TextChanged (изменился текст в поле
// редактирования) обоих компонентов TextBox.
// Сначала надо обычным образом создать функцию обработки события
// TextChanged для компонента textBox1, а затем указать ее в качестве
// обработчика события TextChanged для компонента textBox2
private void textBox1_TextChanged(object sender, EventArgs e)
{
    label3.Text = ""; // очистить поле отображения
    // результата расчета
```

```
if ((textBox1.Text.Length == 0) || (textBox2.Text.Length == 0))
    // В поле редактирования нет данных.
    // Сделать кнопку Расчет недоступной
    button1.Enabled = false;
else
    // Сделать кнопку Расчет доступной
    button1.Enabled = true;
}
```

В листинге 2.7 приведен конструктор формы. Он показывает, как можно выполнить настройку компонентов формы "в коде".

Листинг 2.7. Конструктор класса формы программы "Доход"

```
public Form1()
{
    InitializeComponent();

    // сделать кнопку Расчет недоступной
    button1.Enabled = false;
}
```

Завершение работы над проектом

После того как приложение будет отлажено, можно выполнить его сборку. Для этого в меню **Debug** надо выбрать команду **Build**. В результате описанных действий в подкаталоге bin\Release каталога проекта будет создан выполняемый (exe) файл программы.

Для завершения работы над проектом надо в меню **File** выбрать команду **Close Solution**.

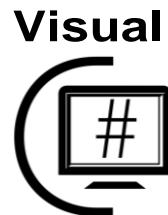
Установка приложения на другой компьютер

В простейшем случае приложение Windows Forms представляет собой один-единственный exe-файл (в .NET-терминологии — сборку). Таким образом, чтобы установить созданное в Microsoft Visual C# приложение на другой компьютер, достаточно перенести (скопировать) на диск этого компьютера exe-файл. Вместе с тем, необходимо помнить: для того чтобы .NET-приложение могло работать на другом компьютере, на нем должна быть установлена платформа .NET Framework соответствующей версии (по умолчанию проекты Visual Studio 2010 компилируются в режиме использования Microsoft .NET Framework 4.0). Если на компьютере пользователя платформа .NET Framework не установлена, то при запуске приложения будет выведено сообщение о необходимости установить .NET Framework (рис. 2.33).



Рис. 2.33. Сообщение об ошибке при попытке запустить приложение на компьютере, на котором нет Microsoft .NET Framework

Профессиональный подход к разработке программного обеспечения предполагает, что программист создает не только приложение, но и установщик — программу, обеспечивающую установку приложения на компьютер пользователя. О том, как создать установщик, рассказывается в *главе 10*.



ГЛАВА 3

Базовые компоненты

Microsoft Visual Studio, являясь средой быстрой разработки, поддерживает технологию визуального проектирования, идея которой заключается в том, что среда разработки предоставляет в распоряжение программиста набор **компонентов**, которые можно поместить на форму.

Компонент — это объект, предназначенный для решения некоторой задачи. Компоненты обеспечивают ввод данных, отображение результатов (такие компоненты называют компонентами пользовательского интерфейса), доступ к базам данных, решение других задач.

В этой главе демонстрируется назначение базовых компонентов, к которым можно отнести компоненты, обеспечивающие взаимодействие с пользователем (`TextBox`, `Label`, `Button` и др.), отображение иллюстраций (`PictureBox`), диалогов (`SaveDialog`, `OpenDialog` и др.) и некоторые другие, например `Timer`.

Компоненты находятся в палитре компонентов (рис. 3.1).

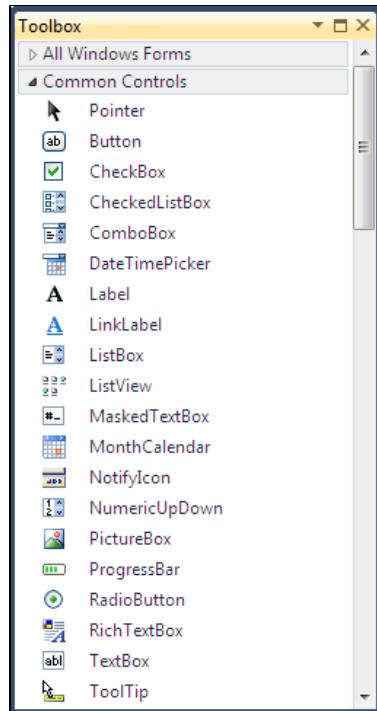


Рис. 3.1. Палитра компонентов

Label

Компонент `Label` (рис. 3.2) предназначен для отображения текстовой информации. Задать текст, отображаемый в поле компонента, можно как во время разработки формы, так и во время работы программы, присвоив значение свойству `Text`.

Свойства компонента приведены в табл. 3.1.

Таблица 3.1. Свойства компонента Label

Свойство	Описание
Name	Имя (идентификатор) компонента. Используется в программе для доступа к компоненту
Text	Отображаемый текст
Location	Положение компонента на поверхности формы
Size	Размер компонента (области отображения текста)
Font	Шрифт, используемый для отображения текста
ForeColor	Цвет текста, отображаемого в поле компонента
BackColor	Цвет закраски области вывода текста
TextAlign	Способ выравнивания (расположения) текста в поле компонента. Всего существует девять способов расположения текста. На практике наиболее часто используют выравнивание по левой и верхней границам (TopLeft), посередине (TopCenter) и по центру (MiddleCenter)
BorderStyle	Вид рамки (границы) компонента. По умолчанию граница вокруг поля Label отсутствует (значение свойства равно None). Граница компонента может быть обычной (Fixed3D) или тонкой (FixedSingle)

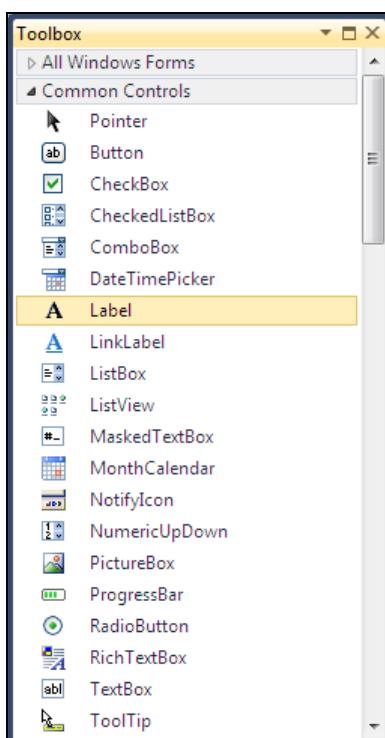


Рис. 3.2. Компонент Label

Чтобы в поле компонента `Label` вывести числовое значение, это значение надо преобразовать в строку. Сделать это можно при помощи метода `ToString`.

Цвет текста (`ForeColor`) и фона (`BackColor`) можно задать, указав название цвета (`Color.Red`, `Color.Blue`, `Color.Green` и т. д.) или элемент цветовой схемы операционной системы (`System.Drawing.SystemColors.Control`, `System.Drawing.SystemColors.ControlText` и т. д.). Во втором случае цвет будет "привязан" к текущей цветовой схеме операционной системы и будет автоматически меняться при каждой ее смене. По умолчанию для элементов управления используется второй способ кодирования цвета. Цвет фона может быть "прозрачным" (`Color.Transparent`).

Программа "Конвертер" (ее форма приведена на рис. 3.3, а текст функции обработки события `Click`, возникающего при щелчке на кнопке **OK**, — в листинге 3.1) демонстрирует возможности компонента `Label`. Она показывает, как во время работы программы изменить цвет текста, отображаемого в поле компонента, как вывести в поле компонента значение переменной. Программа пересчитывает цену из долларов в рубли. Если пользователь оставит какое-либо из полей незаполненным, то в результате щелчка на кнопке **OK** в поле компонента `Label13` красным цветом отображается сообщение об ошибке. Если все поля формы заполнены, то в поле компонента `Label13` отображается результат расчета.

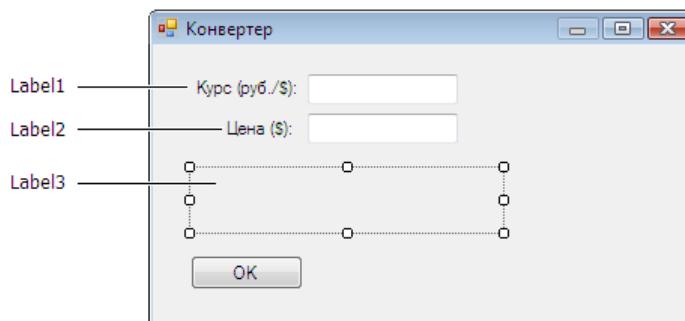


Рис. 3.3. Форма программы "Конвертер"

Листинг 3.1. Обработка события Click

```
private void button1_Click(object sender, EventArgs e)
{
    double usd; // цена в долларах
    double k; // курс

    double rub; // цена в рублях

    if ((textBox1.TextLength != 0) && (textBox2.TextLength != 0))
    {
        usd = Convert.ToDouble(textBox1.Text);
        k = Convert.ToDouble(textBox2.Text);

        rub = usd * k;
        label13.Text = rub.ToString("0.00");
    }
}
```

```

rub = usd * k;

label3.ForeColor = System.Drawing.SystemColors.ControlText;
label3.Text = usd.ToString("n") + "$ = " +
            rub.ToString("C"); // финансовый формат
}

else
{
    label3.ForeColor = Color.Red;
    label3.Text = "Надо ввести данные в оба поля!";
}
}

```

TextBox

Компонент TextBox (рис. 3.4) предназначен для ввода данных с клавиатуры. В зависимости от настройки компонента, в поле редактирования можно ввести одну или несколько строк текста. Свойства компонента приведены в табл. 3.2.

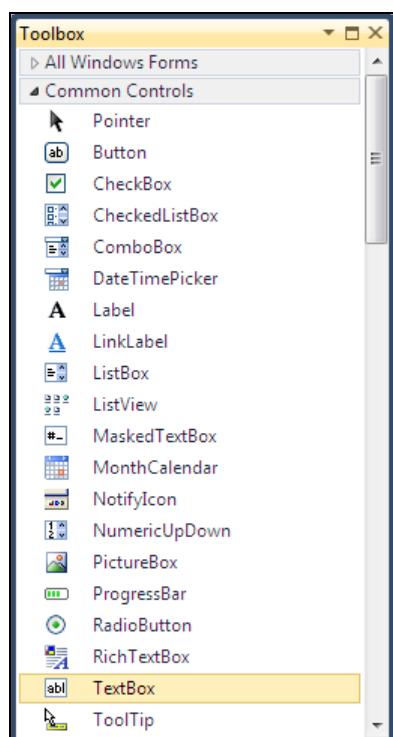


Рис. 3.4. Компонент TextBox

Таблица 3.2. Свойства компонента TextBox

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к компоненту и его свойствам, в том числе к тексту, который находится в поле редактирования

Таблица 3.2 (окончание)

Свойство	Описание
Text	Текст, который находится в поле редактирования
Location	Положение компонента на поверхности формы
Size	Размер компонента
Font	Шрифт, используемый для отображения текста в поле компонента
ForeColor	Цвет текста, находящегося в поле компонента
BackColor	Цвет фона поля компонента
BorderStyle	Вид рамки (границы) компонента. Граница компонента может быть обычной (Fixed3D), тонкой (FixedSingle). Граница вокруг компонента может отсутствовать (в этом случае значение свойства равно None)
TextAlign	Способ выравнивания текста в поле компонента. Текст в поле компонента может быть прижат к левой границе компонента (Left), правой (Right) или находится по центру (Center)
MaxLength	Максимальное количество символов, которое можно ввести в поле компонента
PasswordChar	Символ, который используется для отображения вводимых пользователем символов (введенная пользователем строка находится в свойстве Text)
Multiline	Разрешает (True) или запрещает (False) ввод нескольких строк текста
Dock	Способ привязки положения и размера компонента к размеру формы. По умолчанию привязка отсутствует (None). Если значение свойства равно Top или Bottom, то ширина компонента устанавливается равной ширине формы и компонент прижимается соответственно к верхней или нижней границе формы. Если значение свойства Dock равно Fill, а свойства Multiline — True, то размер компонента устанавливается максимально возможным
Lines	Массив строк, элементы которого содержат текст, находящийся в поле редактирования, если компонент находится в режиме MultiLine. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
SkroolBars	Задает отображаемые полосы прокрутки: Horizontal — горизонтальная; Vertical — вертикальная; Both — горизонтальная и вертикальная; None — не отображать

Если не предпринимать никаких усилий, то в поле компонента TextBox отображаются все символы, которые пользователь набирает на клавиатуре, что не всегда удобно. Программа может обеспечить фильтрацию вводимых символов путем запрета отображения запрещенных символов. Для того чтобы символ не отображался в поле редактирования, в процедуре обработки события KeyPress параметру Handled надо присвоить значение True.

Представленная далее программа (форма приведена на рис. 3.5, а текст — в листинге 3.2) демонстрирует использование компонента TextBox для ввода данных различного типа. Программа спроектирована таким образом, что в режиме ввода текста в поле редактирования можно ввести любой символ, в режиме ввода

целого числа — только цифры. В режиме ввода дробного числа кроме цифр в поле редактирования можно ввести символ-разделитель (запятую или точку, в зависимости от настройки операционной системы). Какой символ правильный — можно узнать, обратившись к объекту `CultureInfo`, который содержит значения настроек операционной системы. Следует обратить внимание, для того чтобы объект `CultureInfo` стал доступен в программу надо добавить ссылку (директиву `using` с указанием пространства имен) на пространство имен `System.Globalization`.

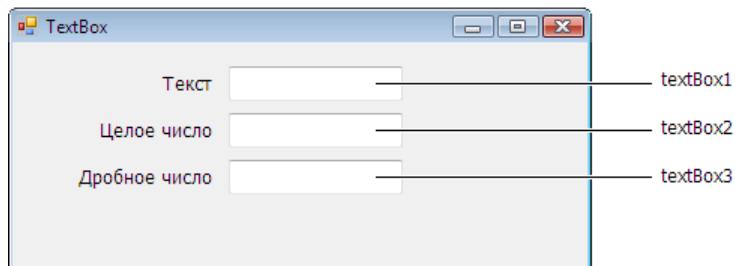


Рис. 3.5. Форма программы TextBox

Листинг 3.2. Обработка события KeyPress компонентов TextBox

```
// изменилось содержимое поля редактирования
private void textBox1_TextChanged(object sender, EventArgs e)
{
    // "стереть" предыдущий результат расчета
    label2.Text = "";

    // Если в поле textBox1 нет данных, сделать кнопку button1 недоступной
    if (textBox1.TextLength == 0)
        button1.Enabled = false;
    else
        button1.Enabled = true;
}

// щелчок на кнопке OK
private void button1_Click(object sender, EventArgs e)
{
    double mile; // расстояние в милях
    double km; // расстояние в километрах

    // Если в поле редактирования нет данных, то при попытке
    // преобразовать пустую строку в число возникает исключение.
    try
    {
        mile = Convert.ToDouble(textBox1.Text);

        km = mile * 1.609344;
    }
}
```

```
label2.Text = km.ToString("n") + " км." // числовой (numeric) формат
}

catch
{
    // обработка исключения:
    // переместить курсор в поле редактирования
    textBox1.Focus();
}

// нажатие клавиши в поле редактирования
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    // Правильными символами считаются цифры, запятая, <Enter> и <Backspace>.
    // Будем считать правильным символом также точку, но заменим ее запятой.
    // Остальные символы запрещены.
    // Чтобы запрещенный символ не отображался в поле редактирования,
    // присвоим значение True свойству Handled параметра e.

    if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
    {
        // цифра
        return;
    }

    char aDecimalSeparator = System.Globalization.CultureInfo.CurrentCulture.
        NumberFormat.NumberDecimalSeparator[0];

    if (e.KeyChar == '.')
    {
        // точку заменим запятой
        e.KeyChar = aDecimalSeparator;
    }

    if (e.KeyChar == aDecimalSeparator)
    {
        if (textBox1.Text.IndexOf(aDecimalSeparator) != -1)
        {
            // запятая уже есть в поле редактирования
            e.Handled = true;
        }
        return;
    }

    if (Char.IsControl (e.KeyChar) )
    {
        // <Enter>, <Backspace>, <Esc>
        if ( e.KeyChar == (char) Keys.Enter)
```

```

    // Нажата клавиша <Enter>.
    // Установить курсор на кнопку OK
    button1.Focus();

    return;
}

// остальные символы запрещены
e.Handled = true;
}

```

Button

Компонент Button (рис. 3.6) представляет собой командную кнопку. Свойства компонента приведены в табл. 3.3.

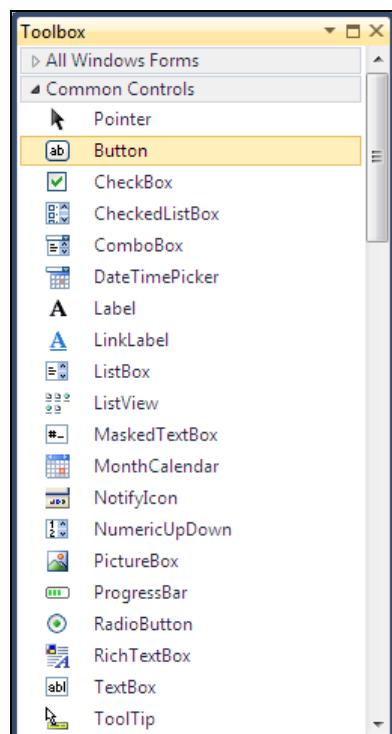


Рис. 3.6. Компонент Button

Таблица 3.3. Свойства компонента Button

Свойство	Описание
Name	Имя компонента. Используется в программе для доступа к свойствам компонента
Text	Текст (надпись) на кнопке
TextAlign	Положение текста (надписи) на кнопке. Надпись может располагаться в центре (MiddleCenter), быть прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения надписи (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)

Таблица 3.3 (окончание)

Свойство	Описание
FlatStyle	Кнопка может быть стандартной (Standard), плоской (Flat) или "всплывающей" (Popup)
Location	Положение кнопки на поверхности формы. Уточняющее свойство X определяет расстояние от левой границы кнопки до левой границы формы, уточняющее свойство Y — от верхней границы кнопки до верхней границы клиентской области формы (нижней границы заголовка)
Size	Размер кнопки
Font	Шрифт, используемый для отображения текста на кнопке
ForeColor	Цвет текста, отображаемого на кнопке
Enabled	Признак доступности кнопки. Кнопка доступна, если значение свойства равно True, и недоступна (например, событие Click в результате щелчка на кнопке не возникает), если значение свойства равно False
Visible	Позволяет скрыть кнопку (False) или сделать ее видимой (True)
Cursor	Вид указателя мыши при позиционировании указателя на кнопке
Image	Картинка на поверхности формы. Рекомендуется использовать gif-файл, в котором определен прозрачный цвет
ImageAlign	Положение картинки на кнопке. Картинка может располагаться в центре (MiddleCenter), быть прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения картинки на кнопке (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
ImageList	Набор картинок, используемых для обозначения различных состояний кнопки. Представляет собой объект типа ImageList. Чтобы задать значение свойства, в форму приложения надо добавить компонент ImageList
ImageIndex	Номер (индекс) картинки из набора ImageList, которая отображается на кнопке
ToolTip	Подсказка, появляющаяся рядом с указателем мыши при позиционировании его на кнопке. Чтобы свойство было доступно, в форму приложения надо добавить компонент ToolTip

Следующая программа (ее форма приведена на рис. 3.7, а текст — в листинге 3.3) демонстрирует использование компонента Button. Программа пересчитывает расстояние из миль в километры. Расчет и отображение результата выполняет процедура обработки события click. Следует обратить внимание, что кнопка **OK** доступна только в том случае, если в поле редактирования находятся данные (хотя бы одна цифра). Управляет доступностью кнопки процедура обработки события TextChanged компонента TextBox. Процедура контролирует количество символов, которое находится в поле редактирования, и, если в поле нет ни одной цифры, присваивает значение False свойству Enabled и, тем самым, делает кнопку недоступной. В процессе создания формы свойству Enabled кнопки надо присвоить значение False.

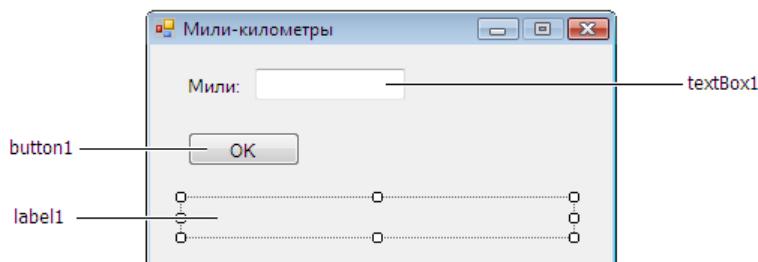


Рис. 3.7. Во время работы программы кнопка **OK** будет доступна только в том случае, если в поле TextBox пользователь введет хотя бы одну цифру

Листинг 3.3. Мили-километры

```
// содержимое поля редактирования изменилось
private: System::Void textBox1_TextChanged(System::Object^ sender,
                                             System::EventArgs^ e)
{
    // Если в поле textBox1 нет данных, сделать кнопку button1 недоступной
    if (textBox1->Text->Length == 0)
        button1->Enabled = false;
    else
        button1->Enabled = true;
}

// нажатие клавиши в поле textBox1
private: System::Void textBox1_KeyPress(System::Object^ sender,
                                         System::Windows::Forms::KeyPressEventArgs^ e)
{
    // Правильными символами считаются цифры, запятая, <Enter> и <Backspace>.
    // Будем считать правильным символом также точку,
    // но заменим ее запятой. Остальные символы запрещены.
    // Чтобы запрещенный символ не отображался в поле редактирования,
    // присвоим значение True свойству Handled параметра e

    if ((e->KeyChar >= '0') && (e->KeyChar <= '9'))
    {
        // цифра
        return;
    }

    if (e->KeyChar == '.')
    {
        // точку заменим запятой
        e->KeyChar = ',';
    }
}
```

```
if (e->KeyChar == ',')
{
    if (textBox1->Text->IndexOf(',') != -1)
    {
        // запятая уже есть в поле редактирования
        e->Handled = true;
    }
    return;
}

if (Char::IsControl(e->KeyChar))
{
    // <Enter>, <Backspace>, <Esc>
    if (e->KeyChar == (char) Keys::Enter)
        // Нажата клавиша <Enter>.
        // Установить "фокус" на кнопку OK
        button1->Focus();
    return;
}

// остальные символы запрещены
e->Handled = true;
}

// щелчок на кнопке OK
private: System::Void button1_Click(System::Object^ sender,
                                      System::EventArgs^ e)
{
    double mile; // расстояние в милях
    double km;   // расстояние в километрах

    mile = Convert::.ToDouble(textBox1->Text);

    km = mile * 1.609344;

    label2->Text = mile.ToString("n") + " miles - " +
                    km.ToString("n") + " км.";
}
```

На кнопке может находиться картинка. Существуют два способа поместить картинку на кнопку. Первый — присвоить значение свойству `Image`. Второй — добавить в форму компонент `ImageList`, выполнить его настройку, установить связь между компонентами `Button` и `ImageList` и затем, присвоив значение свойству `ImageIndex` компонента `Button`, задать картинку для кнопки. Первый способ технически проще, но при его использовании нельзя задать прозрачный цвет, поэтому картинка должна быть прямоугольной или цвет фона изображения должен совпа-

дать с цветом кнопки. При использовании второго способа есть возможность задать прозрачный цвет.

CheckBox

Компонент CheckBox (рис. 3.8) представляет флажок, который может находиться в одном из двух состояний: выбранном или невыбранном. Часто вместо "выбранный" говорят "установленный", а вместо "невыбранный" — "сброшенный" или "выключеный". Рядом с флажком, как правило, находится поясняющий текст. Компоненты CheckBox обычно используют для выбора нескольких опций из ряда возможных.

Свойства компонента CheckBox приведены в табл. 3.4.

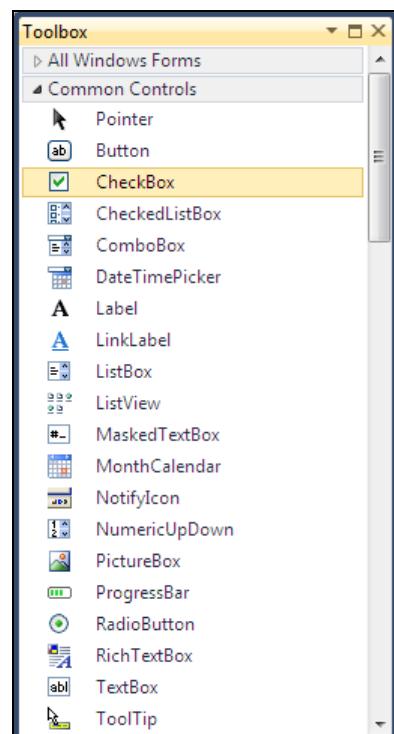


Рис. 3.8. Компонент CheckBox

Таблица 3.4. Свойства компонента CheckBox

Свойство	Описание
Text	Текст, поясняющий назначение флагка
Checked	Состояние (вид) флагка. Если флагок выбран, то значение свойства равно True. Если флагок сброшен, то значение свойства равно False
TextAlign	Положение текста в поле отображения текста. Текст может располагаться в центре поля (MiddleCenter), быть прижат к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения текста надписи (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
CheckAlign	Положение кнопки в поле компонента. Кнопка может быть прижата к левой верхней границе (TopLeft), прижата к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (MiddleLeft). Есть и другие варианты размещения кнопки в поле компонента

Таблица 3.4 (окончание)

Свойство	Описание
Enabled	Свойство позволяет сделать флажок недоступным (False)
Visible	Свойство позволяет скрыть (False) флажок
AutoCheck	Свойство определяет, должно ли автоматически изменяться состояние флажка в результате щелчка на его изображении. По умолчанию значение равно True
FlatStyle	Стиль (вид) флажка. Флажок может быть обычным (Standard), плоским (Flat) или "всплывающим" (PopUp). Стиль определяет поведение флажка при позиционировании указателя мыши на его изображении
Appearance	Определяет вид флажка. Флажок может выглядеть обычным образом (Normal) или как кнопка (Button)
Image	Картина, которая отображается в поле компонента
ImageAlign	Положение картинки в поле компонента. Картина может располагаться в центре (MiddleCenter), быть прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения картинки на кнопке (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
ImageList	Набор картинок, используемых для обозначения различных состояний кнопки. Представляет собой объект типа ImageList. Чтобы задать значение свойства, в форму приложения следует добавить компонент ImageList
ImageIndex	Номер (индекс) картинки из набора ImageList, которая отображается в поле компонента

Состояние флажка изменяется в результате щелчка на его изображении (если значение свойства AutoCheck равно True). При этом возникает событие CheckedChanged, а потом событие Click. Если значение свойства AutoCheck равно False, то в результате щелчка на флагке возникает событие Click, а затем, если процедура обработки этого события изменит состояние кнопки, возникает событие CheckedChanged.

Следующая программа (ее окно приведено на рис. 3.9, а текст — в листинге 3.4) демонстрирует использование компонента CheckBox. Программа позволяет посчитать цену автомобиля в зависимости от выбранной комплектации.

Листинг 3.4. Комплектация (компонент CheckBox)

```
// щелчок на кнопке OK
private void button1_Click(object sender, EventArgs e)
{
    double cena;      // цена в базовой комплектации
    double dop;       // сумма за доп. оборудование
    double discount; // скидка
    double total;     // общая сумма
```

```
cena = 415000;
dop = 0;

if (checkBox1.Checked)
{
    // коврики
    dop += 1200;
}

if (checkBox2.Checked)
{
    // защита картера
    dop += 4500;
}

if (checkBox3.Checked)
{
    // зимние шины
    dop += 12000;
}

if (checkBox4.Checked)
{
    // литые диски
    dop += 16000;
}

total = cena + dop;

System.String st;
st = "Цена в выбранной комплектации: " + total.ToString("C");
if (dop != 0)
{
    st += "\nВ том числе доп. оборудование: " + dop.ToString("C");
}

if ((checkBox1.Checked) && (checkBox2.Checked) &&
    (checkBox3.Checked) && (checkBox4.Checked))
{
    // Скидка предоставляется, если выбраны все опции
    discount = dop * 0.1;
    total = total - discount;
    st += "\nСкидка на доп. оборудование (10%): " +
        discount.ToString("C") +
        "\nИтого: " + total.ToString("C");
}
label2.Text = st;
}
```

```
// Пользователь изменил состояние переключателя.
// Функция обрабатывает событие CheckedChanged
// компонентов checkBox1-checkBox4
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    label2.Text = "";
}
```

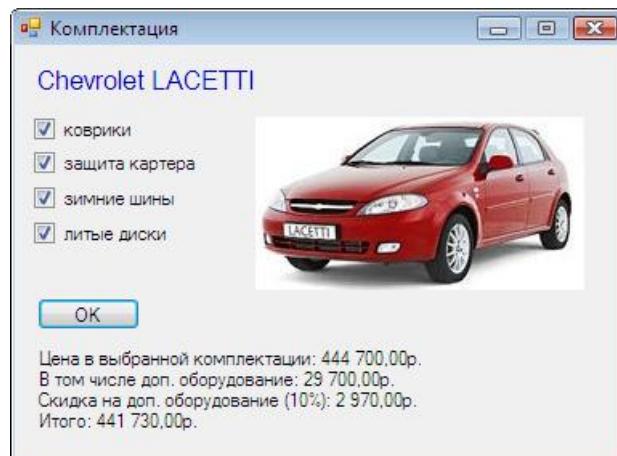


Рис. 3.9. Окно программы "Комплектация"

Для отображения картинки в окне программы "Комплектация" используется компонент PictureBox. Картинка загружается в компонент из файла, который должен находиться в том же каталоге, что и выполняемый файл. Картинка загружается во время работы программы. Делает это конструктор формы (листинг 3.5). Следует обратить внимание на то, что при запуске программы из среды разработки файл иллюстрации должен находиться в подкаталоге bin\Debug каталога проекта. Если картинка находится в папке иллюстраций пользователя (Мои рисунки), то получить путь к файлу иллюстрации можно, вызвав функцию System.Environment.GetFolderPath и указав в качестве ее параметра значение System.Environment.SpecialFolder.MyPictures.

Листинг 3.5. Конструктор формы

```
public Form1()
{
    InitializeComponent();

    pictureBox1.Image =
        Image.FromFile(Application.StartupPath+"\\"+Lacetti_r.jpg");
}
```

RadioButton

Компонент RadioButton (рис. 3.10) представляет собой переключатель, состояние которого зависит от состояния других переключателей (компонентов RadioButton). Обычно компоненты RadioButton объединяют в группу (достигается это путем размещения нескольких компонентов в поле компонента GroupBox). В каждый момент времени только один из переключателей группы может находиться в выбранном состоянии (возможна ситуация, когда ни один из переключателей не выбран). Состояние компонентов, принадлежащих одной группе, не зависит от состояния компонентов, принадлежащих другой группе.

Свойства компонента приведены в табл. 3.5.

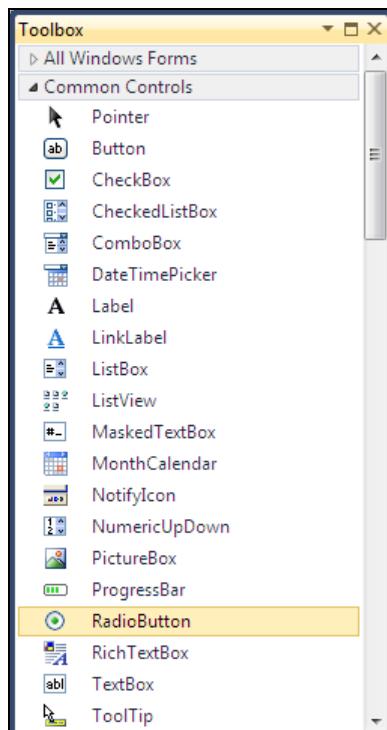


Рис. 3.10. Компонент RadioButton

Таблица 3.5. Свойства компонента RadioButton

Свойство	Описание
Text	Текст, который находится справа от переключателя
Checked	Состояние, внешний вид переключателя. Если переключатель выбран, то значение свойства Checked равно True; если не выбран, то значение свойства Checked равно False
TextAlign	Положение текста в поле отображения текста. Текст может располагаться в центре поля (MiddleCenter), быть прижат к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения текста надписи (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
CheckAlign	Положение кнопки в поле компонента. Кнопка может быть прижата к левой верхней границе (TopLeft), прижата к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (MiddleLeft). Есть и другие варианты размещения кнопки в поле компонента

Таблица 3.5 (окончание)

Свойство	Описание
Enabled	Свойство позволяет сделать переключатель недоступным (False)
Visible	Свойство позволяет скрыть (False) переключатель
AutoCheck	Свойство определяет, должно ли автоматически изменяться состояние переключателя в результате щелчка на его изображении. По умолчанию значение равно True
FlatStyle	Стиль переключателя. Переключатель может быть обычным (Standard), плоским (Flat) или "всплывающим" (PopUp). Стиль переключателя определяет его поведение при позиционировании указателя мыши на изображении кнопки
Appearance	Определяет вид переключателя. Переключатель может выглядеть обычным образом (Normal) или как кнопка (Button)
Image	Картинка, которая отображается в поле компонента
ImageAlign	Положение картинки в поле компонента. Картина может располагаться в центре (MiddleCenter), быть прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения картинки на кнопке (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
ImageList	Набор картинок, используемых для обозначения различных состояний переключателя. Представляет собой объект типа ImageList. Чтобы задать значение свойства, в форму приложения надо добавить компонент ImageList
ImageIndex	Номер (индекс) картинки из набора ImageList, которая отображается в поле компонента

Состояние переключателя изменяется в результате щелчка на его изображении (если значение свойства AutoCheck равно True). При этом возникает событие CheckedChanged, а затем событие Click. Если значение свойства AutoCheck равно False, то в результате щелчка на переключателе возникает событие Click, а затем, если процедура обработки этого события изменит состояние кнопки, возникает событие CheckedChanged.

Программа "Фото" (ее форма приведена на рис. 3.11, а текст — в листинге 3.6) демонстрирует использование компонента RadioButton. Программа вычисляет стоимость заказа печати фотографий, в зависимости от их размера. Значения свойств компонентов RadioButton приведены в табл. 3.6. Следует обратить внимание: на форму сначала надо поместить компонент GroupBox, затем — компоненты RadioButton. Помимо события Click, возникающего при щелчке на кнопке **OK**, в программе обрабатывается событие Click компонентов RadioButton. Функция обработки этого события (одна для всех компонентов) очищает поле отображения результата и устанавливает курсор в поле ввода/редактирования.

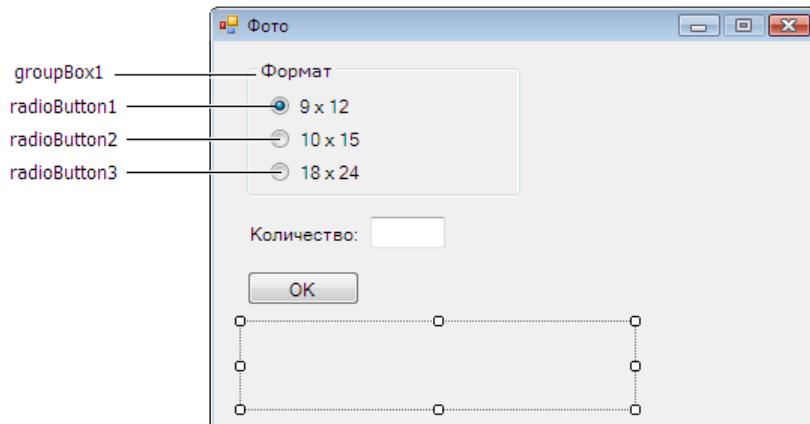


Рис. 3.11. Форма программы "Фото"

Таблица 3.6. Значения свойств компонентов RadioButton

Свойство	Значение
radioButton1.Text	9x12
radioButton1.Checked	true
radioButton2.Text	10x15
radioButton3.Text	18x24

Листинг 3.6. Фото (функции обработки событий)

```
// Обработка события Click компонентов radioButton1 - radioButton3
private void radioButton1_Click(object sender, EventArgs e)
{
    label2.Text = "";
    // установить курсор в поле Количество
    textBox1.Focus();
}

// щелчок на кнопке OK
private void button1_Click(object sender, EventArgs e)
{
    double cena = 0; // цена
    int n;           // количество фотографий
    double sum;      // сумма

    if (radioButton1.Checked)
        cena = 8.50;
    if (radioButton2.Checked)
        cena = 10;
}
```

```
if (radioButton3.Checked)
    cena = 15.5;

n = Convert.ToInt32(textBox1.Text);
sum = n * cena;

label2.Text = "Цена: " + cena.ToString("c") +
    "\nКоличество: " + n.ToString() + "шт.\n" +
    "Сумма заказа: " + sum.ToString("C");
}

// В поле Количество можно ввести только целое число
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
        return;

    if (Char.IsControl(e.KeyChar))
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            // нажата клавиша <Enter>
            button1.Focus();
        }
        return;
    }
    // остальные символы запрещены
    e.Handled = true;
}

// Изменилось содержимое поля редактирования
private void textBox1_TextChanged(object sender, EventArgs e)
{
    if (textBox1.Text.Length == 0)
        button1.Enabled = false;
    else
        button1.Enabled = true;

    label2.Text = "";
}
```

GroupBox

Компонент **GroupBox** (рис. 3.12) представляет собой контейнер для других компонентов. Обычно он используется для объединения в группы компонентов **RadioButton** по функциональному признаку.

Свойства компонента *GroupBox* приведены в табл. 3.7.

Таблица 3.7. Свойства компонента *GroupBox*

Свойство	Описание
Text	Заголовок — текст, поясняющий назначение компонентов, которые находятся в поле компонента <i>GroupBox</i>
Enabled	Позволяет управлять доступом к компонентам, находящимся в поле (на поверхности) компонента <i>GroupBox</i> . Если значение свойства равно <i>False</i> , то все находящиеся в поле <i>GroupBox</i> компоненты недоступны
Visible	Позволяет скрыть (сделать невидимым) компонент <i>GroupBox</i> и все компоненты, которые находятся на его поверхности

Следующая программа (ее форма приведена на рис. 3.13, а текст — в листинге 3.7) демонстрирует использование компонентов *GroupBox* и *RadioButton*. Компоненты *RadioButton* принадлежат к двум разным группам (находятся в полях разных компонентов *GroupBox*). Это позволяет установить в выбранное состояние два переключателя одновременно, по одному в каждой группе. Значения свойств компонентов *GroupBox* и *RadioButton* приведены в табл. 3.8.

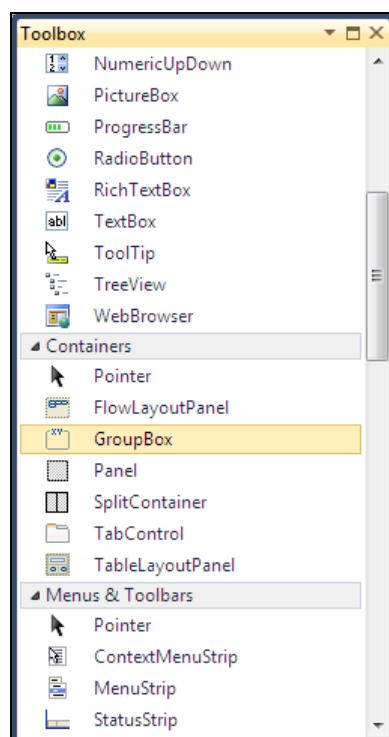


Рис. 3.12. Компонент *GroupBox*

Таблица 3.8. Значения свойств компонентов

Компонент	Свойство	Значение
groupBox1	Text	Формат
radioButton1	Text	9x12
	Checked	true

Таблица 3.8 (окончание)

Компонент	Свойство	Значение
radioButton2	Text	10x15
radioButton3	Text	18x24
groupBox2	Text	Бумага
radioButton4	Text	глянцевая
	Checked	true
radioButton5	Text	матовая

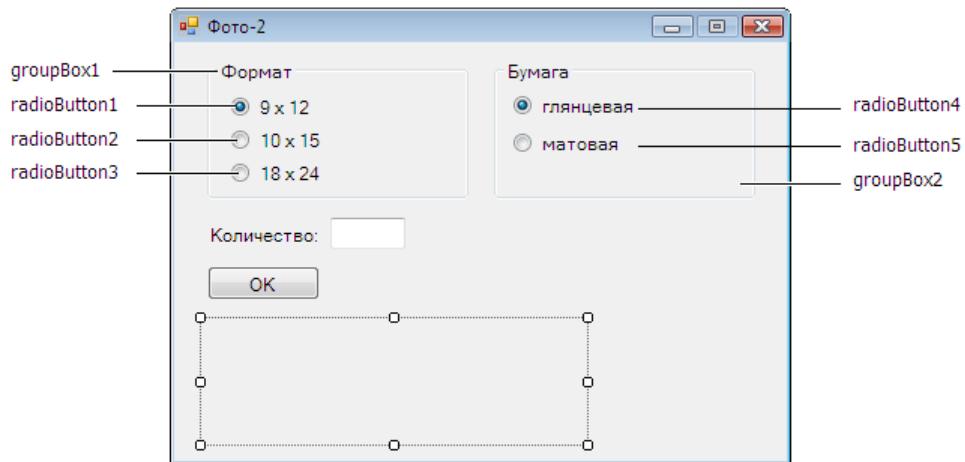


Рис. 3.13. Форма программы "Фото-2"

Листинг 3.7. Фото-2 (компоненты GroupBox И RadioButton)

```
// Обработка события Click компонентов radioButton1 - radioButton5
private void radioButton1_Click(object sender, EventArgs e)
{
    label2.Text = "";
    // установить курсор в поле Количество
    textBox1.Focus();
}

// щелчок на кнопке OK
private void button1_Click(object sender, EventArgs e)
{
    double cena = 0; // цена
    int n;           // количество фотографий
    double sum;      // сумма
```

```
string format = "";
string paper = "";

if (radioButton1.Checked)
{
    cena = 8.50;
    format = "9x12";
}

if (radioButton2.Checked)
{
    cena = 10;
    format = "10x15";

}

if (radioButton3.Checked)
{
    cena = 15.5;
    format = "18x24";
}

// бумага
if (radioButton4.Checked)
{
    paper = "глянцевая";
}

if (radioButton5.Checked)
{
    format = "матовая";
    cena = cena + 0.5;
}

n = Convert.ToInt32(textBox1.Text);
sum = n * cena;

label2.Text = "Формат: " + format +
    "\nБумага: " + paper +
    "\nЦена: " + cena.ToString("c") +
    "\nКоличество: " + n.ToString() + "шт.\n" +
    "Сумма заказа: " + sum.ToString("C");
}

// В поле Количество можно ввести только целое число
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
```

```

{
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
        return;

    if (Char.IsControl(e.KeyChar))
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            // нажата клавиша <Enter>
            button1.Focus();
        }
        return;
    }
    // остальные символы запрещены
    e.Handled = true;
}

private void textBox1_TextChanged(object sender, EventArgs e)
{
    if (textBox1.Text.Length == 0)
        button1.Enabled = false;
    else
        button1.Enabled = true;

    label2.Text = "";
}

```

ComboBox

Компонент **ComboBox** (рис. 3.14) представляет собой комбинацию поля редактирования и списка, что позволяет вводить данные путем набора на клавиатуре или выбора значения в списке.

Свойства компонента приведены в табл. 3.9.

Таблица 3.9. Свойства компонента ComboBox

Свойство	Описание
DropDownStyle	Вид компонента: DropDown — поле ввода и раскрывающийся список; Simple — поле ввода и список; DropDownList — раскрывающийся список
Text	Текст, находящийся в поле ввода/редактирования (для компонентов типа DropDown и Simple)
Items	Элементы списка — коллекция строк
Items.Count	Количество элементов списка
Items.SelectedIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно -1

Таблица 3.9 (окончание)

Свойство	Описание
Sorted	Признак необходимости автоматической сортировки (True) списка после добавления очередного элемента
MaxDropDownItems	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше чем MaxDropDownItems, то появляется вертикальная полоса прокрутки
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа DropDownList) или с учетом (для компонента типа Simple) размера области списка или области ввода
DropDownWidth	Ширина области списка
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

Список, отображаемый в поле компонента, можно сформировать во время создания формы или во время работы программы. Чтобы сформировать список во время создания формы, надо в окне **Properties** выбрать свойство **Items**, щелкнуть на кнопке с тремя точками (она находится поле значения строке свойства **Items**) и в появившемся окне **String Collection Editor** (рис. 3.15) ввести элементы списка.

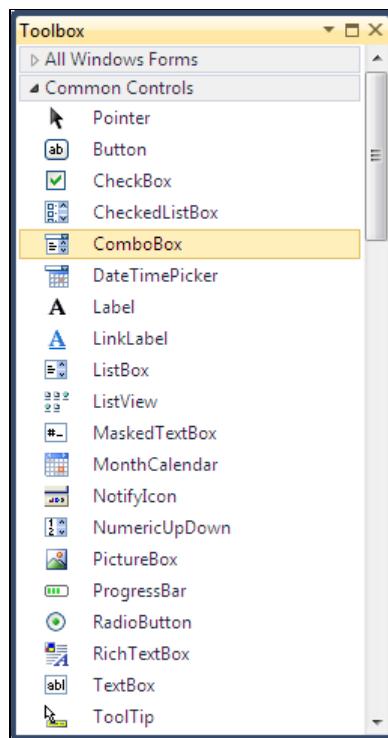


Рис. 3.14. Компонент ComboBox

Чтобы сформировать список во время работы программы, надо применить метод **Add** к свойству **Items**. Например, следующий фрагмент кода формирует упорядоченный по алфавиту список.

```
comboBox1->Items->Add ("пластик");
comboBox1->Items->Add ("алюминий");
comboBox1->Items->Add ("соломка");
```

```
comboBox1->Items->Add ("текстиль");
comboBox1->Items->Add ("бамбук");
comboBox1->Sorted = true;
```

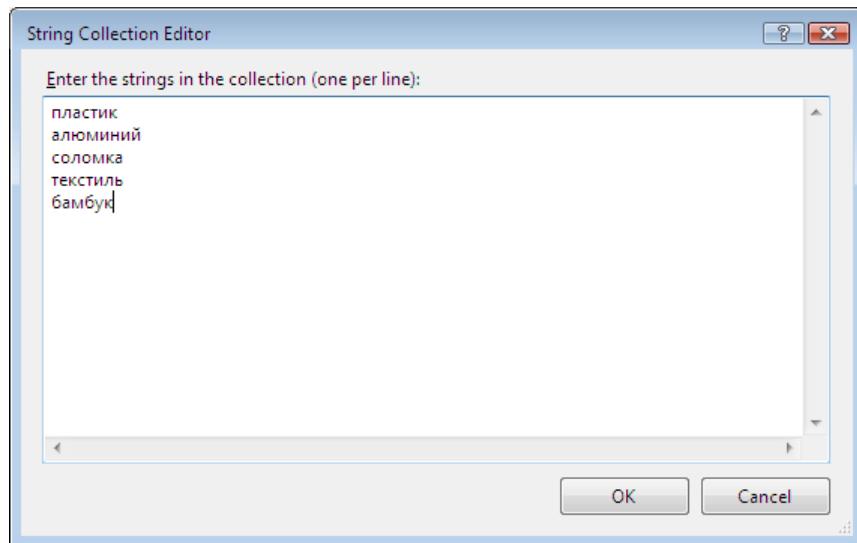


Рис. 3.15. Формирование списка компонента ComboBox во время создания формы

Программа "Жалюзи" демонстрирует использование компонента ComboBox для ввода данных. Форма программы приведена на рис. 3.16, текст — в листинге 3.8. Настройку компонента ComboBox выполняет конструктор формы — свойству SelectedIndex присваивает значение "минус один" (значения свойств компонента ComboBox приведены в табл. 3.10). Поэтому при появлении формы на экране название материала не отображается (ни один из элементов списка не выбран). Кроме того, кнопка **OK** становится доступной только после выбора материала и ввода размеров жалюзи. Доступностью кнопки управляет функция обработки событияTextChanged полей редактирования (одна функция обрабатывает событиеTextChanged обоих компонентов).

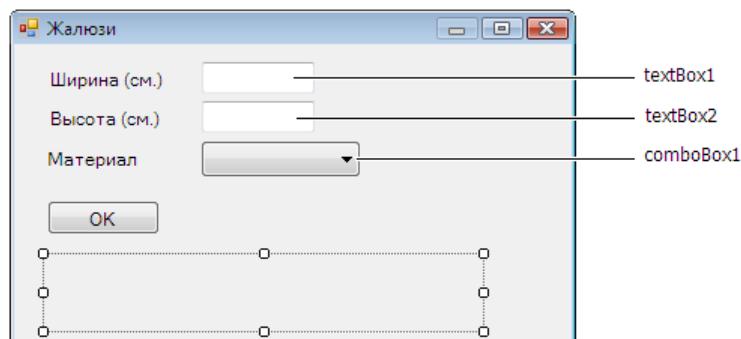


Рис. 3.16. Форма программы "Жалюзи"

Таблица 3.10. Значения свойств компонента ComboBox

Свойство	Значение
comboBox3.DropDownStyle	DropDownList
comboBox3.Sorted	false

Листинг 3.8. Жалюзи (компонент ComboBox)

```
// конструктор
public Form1()
{
    InitializeComponent();

    // настройка компонентов
    comboBox1.DropDownStyle = ComboBoxStyle.DropDownList;

    // comboBox1.Items.Add("пластик");
    // comboBox1.Items.Add("алюминий");
    // comboBox1.Items.Add("бамбук");
    // comboBox1.Items.Add("соломка");
    // comboBox1.Items.Add("текстиль");

    comboBox1.SelectedIndex = 0;
}

// Нажатие клавиши в поле редактирования.
// Функция обрабатывает событие KeyPress компонентов textBox1 и textBox2
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    {
        if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
            return;

        if (Char.IsControl(e.KeyChar))
        {
            if (e.KeyChar == (char)Keys.Enter)
            {
                if (sender.Equals(textBox1))
                    // клавиша <Enter> нажата в поле Ширина
                    // Переместить курсор в поле Высота
                    textBox2.Focus();
                else
                    // Клавиша <Enter> нажата в поле Высота.
                    // Переместить курсор в поле компонента ComboBox
                    comboBox1.Focus();
            }
        }
    }
}
```

```
        return;
    }

    // остальные символы запрещены
    e.Handled = true;
}

}

private void textBox1_TextChanged(object sender, EventArgs e)
{
    if ((textBox1.Text.Length == 0) || (textBox2.Text.Length == 0))
        button1.Enabled = false;
    else
        button1.Enabled = true;

    label4.Text = "";
}

// щелчок на кнопке OK
private void button1_Click(object sender, EventArgs e)
{
    double w;
    double h;
    double cena = 0; // цена за 1 кв. м.
    double sum;

    w = Convert.ToDouble(textBox1.Text);
    h = Convert.ToDouble(textBox2.Text);

    switch (comboBox1.SelectedIndex)
    {
        case 0: cena = 50; break; // пластик
        case 1: cena = 100; break; // алюминий
        case 2: cena = 75; break; // бамбук
        case 3: cena = 70; break; // соломка
        case 4: cena = 60; break; // текстиль
    }

    sum = (w * h) / 10000 * cena;
    label4.Text = "Размер: " + w + "x" + h + " см.\n" +
                  "Цена (р./м.кв.): " + cena.ToString("c") +
                  "\nСумма: " + sum.ToString("c");
}

// в списке Материал пользователь выбрал другой элемент
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    label4.Text = "";
}
```

PictureBox

Компонент PictureBox (рис. 3.17) обеспечивает отображение иллюстрации (рисунка, фотографии и т. п.). Свойства компонента приведены в табл. 3.11.

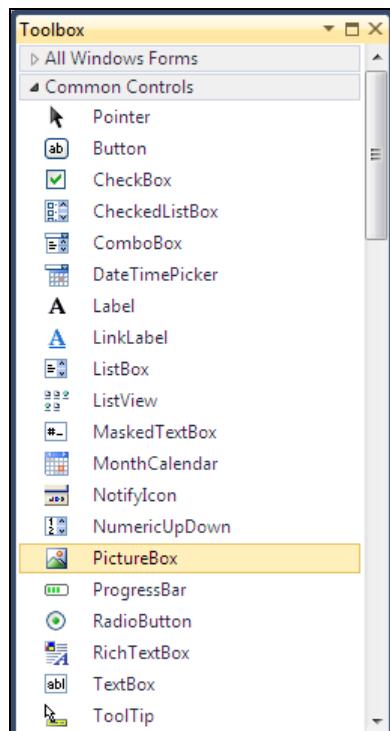


Рис. 3.17. Компонент PictureBox

Таблица 3.11. Свойства компонента PictureBox

Свойство	Описание
Image	Иллюстрация, которая отображается в поле компонента
SizeMode	Режим отображения иллюстрации (способ масштабирования), если размер иллюстрации не соответствует размеру компонента: <ul style="list-style-type: none"> • Normal — масштабирование не выполняется (если размер компонента меньше размера иллюстрации, то отображается только часть иллюстрации); • StretchImage — выполняется масштабирование иллюстрации таким образом, что она занимает всю область отображения (если размер компонента не пропорционален размеру иллюстрации, она будет искажена); • AutoSize — размер компонента автоматически изменяется и соответствует размеру иллюстрации; • CenterImage — центрирование иллюстрации в поле компонента, если размер иллюстрации меньше размера области отображения; • Zoom — выполняется масштабирование так, чтобы иллюстрация занимала область компонента "по максимуму" и при этом отображалась без искажения

Таблица 3.11 (окончание)

Свойство	Описание
Location	Положение компонента (области отображения иллюстрации) на поверхности формы. Уточняющее свойство X определяет расстояние от левой границы области до левой границы формы, уточняющее свойство Y — от верхней границы области до верхней границы клиентской области формы (нижней границы заголовка)
Dock	Способ привязки компонента к компоненту-родителю. Компонент может быть привязан к левой границе (Left), правой (Right), верхней (Top), нижней (Bottom) или занимать всю свободную область компонента-родителя (Fill)
Size	Размер компонента (области отображения иллюстрации). Уточняющее свойство Width определяет ширину области, Height — высоту
Visible	Признак указывает, отображается ли компонент и, соответственно, иллюстрация на поверхности формы
BorderStyle	Вид границы компонента: None — граница не отображается; FixedSingle — тонкая; Fixed3D — объемная
Graphics	Поверхность, на которую можно выводить графику (доступ к графической поверхности компонента есть у процедуры обработки события Paint)

Иллюстрацию, отображаемую в поле компонента PictureBox, можно задать во время разработки формы или загрузить из файла во время работы программы. Чтобы задать иллюстрацию во время создания формы, надо в строке свойства Image щелкнуть на кнопке с тремя точками и в появившемся стандартном окне **Открыть** выбрать файл иллюстрации. Среда разработки создаст битовый образ иллюстрации и поместит его в файл ресурсов (таким образом, в дальнейшем файл иллюстрации программе будет не нужен). Загрузку иллюстрации из файла во время работы программы обеспечивает метод FromFile. В качестве параметра метода надо указать имя файла иллюстрации. Например, инструкция

```
pictureBox1.Image =
    System.Drawing.Bitmap.FromFile("d:\\Photo\\Pict0025.jpg");
```

обеспечивает загрузку и отображение иллюстрации, которая находится в файле d:\\Photo\\Pict0025.jpg. Метод FromFile позволяет работать с файлами BMP, JPEG, GIF, PNG и других форматов.

Программа "Слайд-шоу" демонстрирует использование компонента PictureBox. Окно программы приведено на рис. 3.18. Следует обратить внимание: сначала на форму надо поместить компонент Panel и присвоить его свойству Dock значение Bottom. Затем на форму надо поместить компонент PictureBox и настроить его. Значения свойств компонентов Panel и PictureBox приведены в табл. 3.12. Объявление класса (конструктор, функции обработки событий, функции-члены класса) приведены в листинге 3.9. В начале работы программы конструктор формирует список иллюстраций (jpg-файлов), находящихся в папке пользователя "Мои рисунки". Список хранится в переменной imgList, представляющей собой указатель на список строк. Непосредственное формирование списка выполняет функция FillList. От-

бражение иллюстрации обеспечивает функция `ShowPicture`, которая загружает в компонент `PictureBox` иллюстрацию и в зависимости от ее размера устанавливает значение свойства `SizeMode`. Функция обработки события `click`, возникающего при щелчке на кнопке **Папка**, активизирует отображение стандартного окна **Обзор папок**.

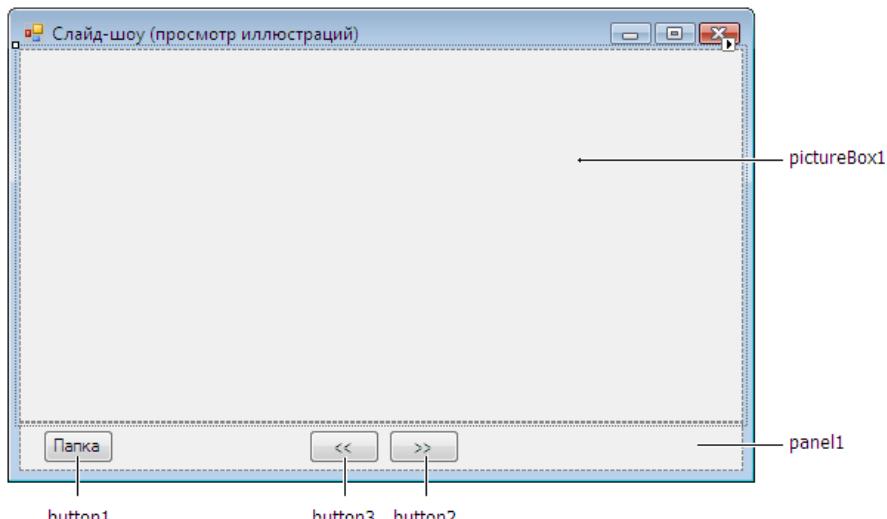


Рис. 3.18. Форма программы "Слайд-шоу"

Таблица 3.12. Значения свойств компонентов

Компонент	Свойство	Значение
panel1	Size.Height	42
	Dock	Bottom
pictureBox1	Dock	Fill
	SizeMode	Zoom

Листинг 3.9. Слайд-шоу (компонент PictureBox)

```
public partial class Form1 : Form
{
    // список jpg-файлов
    List<string> imgList = new List<string>();
    int nImg = 0;      // номер (в списке) отображаемой иллюстрации

    string aPath;      // путь к файлам

    public Form1()
    {
        InitializeComponent();
    }
```

```
DirectoryInfo di; // каталог
// получить имя каталога "Мои рисунки"
di = new DirectoryInfo(Environment.GetFolderPath(
    Environment.SpecialFolder.MyPictures));
aPath = di.FullName;

// сформировать список иллюстраций
FillListBox(aPath);
}

// формирует список иллюстраций aPath - путь к файлам иллюстраций
private Boolean FillListBox(string aPath)
{
    // информация о каталоге
    DirectoryInfo di = new DirectoryInfo(aPath);

    // информация о файлах
    FileInfo[] fi = di.GetFiles("*.jpg");

    // очистить список иллюстраций
    imgList.Clear();

    // добавляем в imgList имена jpg-файлов каталога aPath
    foreach (FileInfo fc in fi)
    {
        imgList.Add(fc.Name);
    }

    if (fi.Length == 0)
    {
        button2.Enabled = false;
        button3.Enabled = false;
        return false;
    }
    else
    {
        nImg = 0;
        ShowPicture(aPath + "\\" + imgList[nImg]);

        // сделать недоступной кнопку Предыдущая
        button2.Enabled = false;

        // если в каталоге один jpg-файл,
        // сделать недоступной кнопку Следующая
        if (imgList.Count == 1)
            button3.Enabled = false;
    }
}
```

```
    else
        button3.Enabled = true;
```

```
    return true;
}
```

```
}
```

```
// выводит иллюстрацию в поле компонента pictureBox1
private void ShowPicture(string aPicture)
{
    pictureBox1.Visible = false;
```

```
// загружаем изображение в pictureBox1
pictureBox1.Image = System.Drawing.Bitmap.FromFile(aPicture);
```

```
// надо масштабировать?
if ((pictureBox1.Image.Width > pictureBox1.Width) ||
    (pictureBox1.Image.Height > pictureBox1.Height))
{
    // да
    pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;
}
else
    // нет, не надо масштабировать
    pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;
```

```
pictureBox1.Visible = true;
```

```
this.Text = aPicture;
}
```

```
// предыдущая картинка
private void button2_Click(object sender, EventArgs e)
{
    // если кнопка "Следующая" недоступна, сделаем ее доступной
    if (!button3.Enabled)
        button3.Enabled = true;
```

```
if (nImg > 0)
{
    nImg--;
    ShowPicture(aPath + "\\" + imgList[nImg]);
```

```
// отображается первая иллюстрация
if (nImg == 0)
{
```

```
// теперь кнопка Предыдущая недоступна
button2.Enabled = false;
}
}

// следующая картинка
private void button3_Click(object sender, EventArgs e)
{
    if (!button2.Enabled)
        button2.Enabled = true;

    if (nImg < imgList.Count)
    {
        nImg++;
        ShowPicture(aPath + "\\" + imgList[nImg]);
        if (nImg == imgList.Count-1)
        {
            button3.Enabled = false;
        }
    }
}

// щелчок на кнопке Обзор
private void button1_Click(object sender, EventArgs e)
{
    // FolderBrowserDialog - окно Обзор папок
    FolderBrowserDialog fb = new FolderBrowserDialog();

    fb.Description =
        "Выберите папку, в которой находятся иллюстрации";

    // кнопка Создать папку недоступна
    fb.ShowNewFolderButton = false;

    // "стартовая" папка
    fb.SelectedPath = aPath;

    // отображаем диалоговое окно
    if (fb.ShowDialog() == DialogResult.OK)
    {
        // пользователь выбрал каталог и щелкнул на кнопке OK
        aPath = fb.SelectedPath;

        if (!FillListBox(fb.SelectedPath))
            // в каталоге нет файлов иллюстраций
            pictureBox1.Image = null;
    }
}
```

ListBox

Компонент `ListBox` (рис. 3.19) представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 3.13.

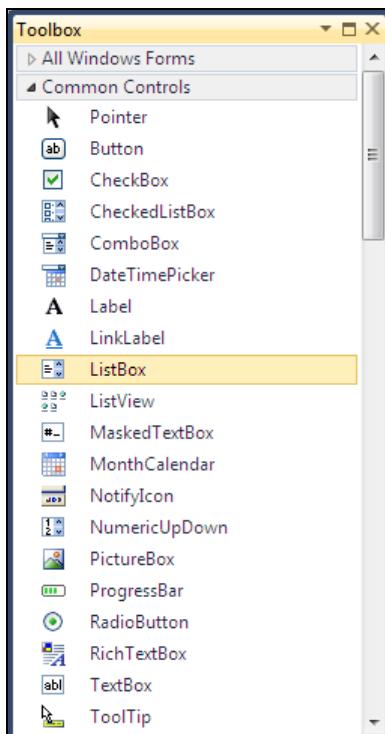


Рис. 3.19. Компонент `ListBox`

Таблица 3.13. Свойства компонента `ListBox`

Свойство	Описание
<code>Items</code>	Элементы списка — коллекция строк
<code>Items.Count</code>	Количество элементов списка
<code>Items.SelectedIndex</code>	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно <code>-1</code>
<code>Sorted</code>	Признак необходимости автоматической сортировки (<code>True</code>) списка после добавления очередного элемента
<code>SelectionMode</code>	Определяет режим выбора элементов списка: <code>One</code> — только один элемент; <code>MultiSimple</code> — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка; <code>MultiExtended</code> — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка при нажатой клавише <code><Ctrl></code> , или выделить диапазон, щелкнув при нажатой клавише <code><Shift></code> на первом и последнем элементе диапазона
<code>ScrollAlwaysVisible</code>	Признак необходимости всегда отображать вертикальную полосу прокрутки. Если значение свойства равно <code>False</code> , то полоса прокрутки отображается, только если все элементы списка нельзя отобразить при заданном размере компонента
<code>MultiColumn</code>	Признак необходимости отображать список в несколько колонок. Количество отображаемых колонок зависит от количества элементов и размера области отображения списка

Таблица 3.13 (окончание)

Свойство	Описание
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа <code>DropDown</code> и <code>DropDownList</code>) или с учетом (для компонента типа <code>Simple</code>) размера области списка или области ввода
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

Список, отображаемый в поле редактирования, можно сформировать во время создания формы или во время работы программы. Чтобы сформировать список во время создания формы, надо щелкнуть на кнопке с тремя точками в строке свойства `Items` и в окне **String Collection Editor** ввести элементы списка. Формирование списка во время работы программы обеспечивает метод `Add` свойства `Items`.

Программа "Просмотр иллюстраций" (вид ее окна приведен на рис. 3.20) демонстрирует использование компонента `ListBox`. Программа позволяет просмотреть иллюстрации (фотографии), находящиеся в указанной пользователем папке.

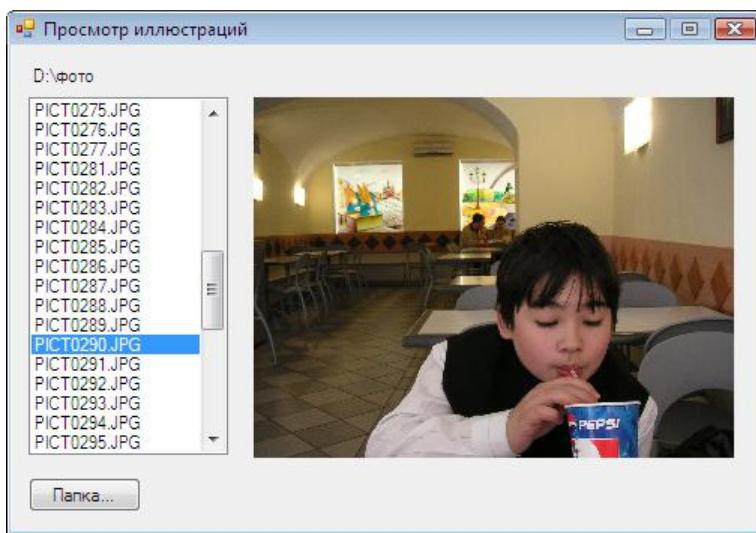


Рис. 3.20. Окно программы "Просмотр иллюстраций"

Текст программы приведен в листинге 3.10. Заполняет список компонента `ListBox` функция `FillListBox`. В начале работы программы эту функцию вызывает конструктор формы. Во время работы — функция обработки события `Click` на кнопке **Папка**. Сначала эта функция отображает диалог **Выбор папки** (отображение диалога обеспечивает метод `ShowDialog` объекта `FolderBrowserDialog`), затем — вызывает функцию `FillListBox`. Для доступа к папке используется объект `DirectoryInfo`. Список файлов представлен в виде массива типа `FileInfo`. Формирует массив метод `GetFiles`. Отображение иллюстраций обеспечивает компонент

PictureBox. Чтобы иллюстрации отображались без искажения, его свойству `SizeMode` надо присвоить значение `Zoom`. Отображение выбранной в списке иллюстрации осуществляется функция обработки события `SelectedIndexChanged`, которое происходит в результате щелчка на элементе списка или перемещения указателя текущего элемента списка при помощи клавиш управления курсором.

Листинг 3.10. Просмотр иллюстраций

```
string aPath; // путь к файлам картинок

public Form1()
{
    InitializeComponent();

    // элементы listBox1 сортируются в алфавитном порядке
    listBox1.Sorted = true;

    DirectoryInfo di; // каталог
    // получить имя каталога "Мои рисунки"
    di = new DirectoryInfo(Environment.GetFolderPath(Environment.SpecialFolder.MyPictures));
    aPath = di.FullName;
    label1.Text = aPath;

    // сформировать список иллюстраций
    FillListBox(aPath);
}

// Формирует список иллюстраций.
// aPath - путь к файлам иллюстраций
private Boolean FillListBox(string aPath)
{
    // информация о каталоге
    DirectoryInfo di = new DirectoryInfo(aPath);

    // информация о файлах
    FileInfo[] fi = di.GetFiles("*.jpg");

    // очистить список listBox
    listBox1.Items.Clear();

    // добавляем в listBox1 имена jpg-файлов, содержащихся в каталоге aPath
    foreach (FileInfo fc in fi)
    {
        listBox1.Items.Add(fc.Name);
    }
}
```

```
label1.Text = aPath;

if (fi.Length == 0) return false;
else
{
    // выбираем первый файл из полученного списка
    listBox1.SelectedIndex = 0;
    return true;
}
}

// Пользователь выбрал другой элемент списка щелчком кнопкой
// мыши или перемещением по списку при помощи клавиатуры
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    pictureBox1.Visible = false;

    // загружаем изображение в pictureBox1
    pictureBox1.Image = new Bitmap( aPath + "\\\" +
        listBox1.SelectedItem.ToString());

    // масштабируем, если нужно
    if ((pictureBox1.Image.Width > pictureBox1.Width) ||
        (pictureBox1.Image.Height > pictureBox1.Height))
    {
        // масштабируем
        pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;
    }

    else
        // Разместить картинку в центре области отображения иллюстраций
        pictureBox1.SizeMode = PictureBoxSizeMode.CenterImage;

    pictureBox1.Visible = true;
}

// щелчок на кнопке Обзор
private void button1_Click(object sender, EventArgs e)
{
    // FolderBrowserDialog - окно Обзор папок
    FolderBrowserDialog fb = new FolderBrowserDialog();

    fb.Description =
        "Выберите папку, \n" +"в которой находятся иллюстрации";
    fb.ShowNewFolderButton = false;

    // отображаем диалоговое окно
    if (fb.ShowDialog() == DialogResult.OK)
```

```

{
    // пользователь выбрал каталог и щелкнул на кнопке OK
    aPath = fb.SelectedPath;
    label1.Text = aPath;

    if (! FillListBox(fb.SelectedPath))
        // в каталоге нет файлов или иллюстраций
        pictureBox1.Image = null;
}
}

```

ListView

Компонент *ListView* (рис. 3.21) предназначен для наглядного представления списков. Свойства компонента приведены в табл. 3.14.

Таблица 3.14. Свойства компонента *ListView*

Свойство	Описание
Columns	Коллекция элементов <i>columnHeader</i> — столбцы, в которых отображается содержимое компонентов элементов списка. Уточняющее свойство <i>Text</i> задает текст в заголовке столбца, свойство <i>Width</i> — ширину столбца
Items	Элементы списка. Коллекция элементов <i>Items</i> определяет содержимое первого столбца. Содержимое остальных столбцов определяется значением элементов коллекции <i>SubItems</i> соответствующего элемента <i>Items</i>
Items.Count	Количество элементов списка
Sorting	Задает правило сортировки элементов списка: <i>None</i> — без сортировки (элементы списка отображаются в том порядке, в котором список был сформирован); <i>Ascending</i> — по возрастанию; <i>Descending</i> — по убыванию
MultiSelect	Определяет режим выбора элементов списка: <i>True</i> — можно выбрать несколько элементов; <i>False</i> — только один элемент. Чтобы выбрать несколько элементов, надо сделать щелчок на нужных элементах списка при нажатой клавише <Ctrl> или выделить диапазон, щелкнув при нажатой клавише <Shift> на первом и последнем элементе диапазона
Scrollable	Признак, позволяющий в случае необходимости отображать полосы прокрутки

Программа "Финансовый калькулятор" (ее форма приведена на рис. 3.22) демонстрирует использование компонента *ListView*. Программа позволяет рассчитать платежи по кредиту (равными долями с начислением процента на текущий остаток долга). Значения свойств компонента *ListView* приведены в табл. 3.15. Список (график платежей по кредиту) формирует функция обработки события *Click*,

возникающего при щелчке на кнопке **OK**. Она и конструктор формы, который выполняет настройку компонента `ListView`, приведены в листинге 3.11. Пример работы программы приведен на рис. 3.23.

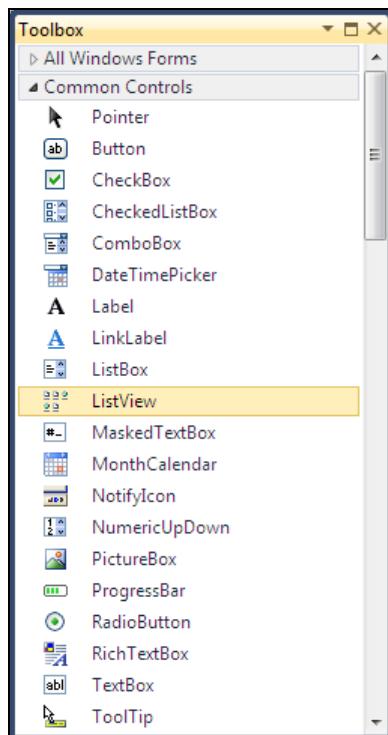
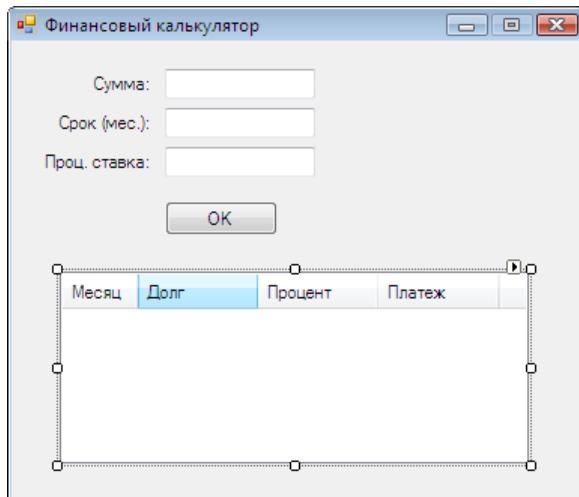
Рис. 3.21. Компонент `ListView`

Рис. 3.22. Форма программы "Финансовый калькулятор"

Таблица 3.15. Значения свойств компонента `ListView`

Компонент	Значение
<code>View</code>	<code>Details</code>
<code>Columns[0].Text</code>	Месяц
<code>Columns[0].Width</code>	50
<code>Columns[1].Text</code>	Долг
<code>Columns[1].Width</code>	80
<code>Columns[2].Text</code>	Процент
<code>Columns[2].Width</code>	80
<code>Columns[3].Text</code>	Платеж
<code>Columns[3].Width</code>	80

Листинг 3.11. Финансовый калькулятор

```
// конструктор формы
public Form1()
{
    InitializeComponent();
    // Настройка listView1 - увеличить ширину компонента
    // на ширину вертикальной полосы прокрутки
    int w = 0;
    for (int i=0; i < listView1.Columns.Count; i++)
        w += listView1.Columns[i].Width;
    if (listView1.BorderStyle == BorderStyle.FixedSingle)
        w +=4;
    listView1.Width = w + 17;

    listView1.FullRowSelect = true;
}

private void button1_Click(object sender, EventArgs e)
{
    float value; // сумма кредита
    int period; // срок
    float rate; // процентная ставка

    int month; // месяц платежа
    float debt; // долг, на начало текущего месяца
    float interest; // плата за кредит (проценты на долг)
    float paying; // сумма платежа
    float suminterest; // общая плата за кредит

    // очистить список - удалить элементы из списка
    listView1.Items.Clear();

    // сумма
    value = System.Convert.ToSingle(textBox1.Text);

    // срок
    period = System.Convert.ToInt32(textBox2.Text);

    // процентная ставка
    rate = System.Convert.ToSingle(textBox3.Text);

    month = 1;
    debt = value; // долг на начало первого месяца
    suminterest = 0;

    // расчет для каждого месяца
    for (int i = 0; i < period; i++)
```

```

{
    interest = debt * (rate/12/100);
    suminterest += interest;
    paying = value/period + interest;

    // добавить в listView1 элемент - строку (данные в первый столбец)
    listView1.Items.Add(month.ToString());

    // добавить в добавленную строку подэлементы -
    // данные во второй, третий и четвертый столбцы
    listView1.Items[i].SubItems.Add(debt.ToString("c"));
    listView1.Items[i].SubItems.Add(interest.ToString("c"));
    listView1.Items[i].SubItems.Add(paying.ToString("c"));

    month++;
    debt = debt - value/period;
}
}

```

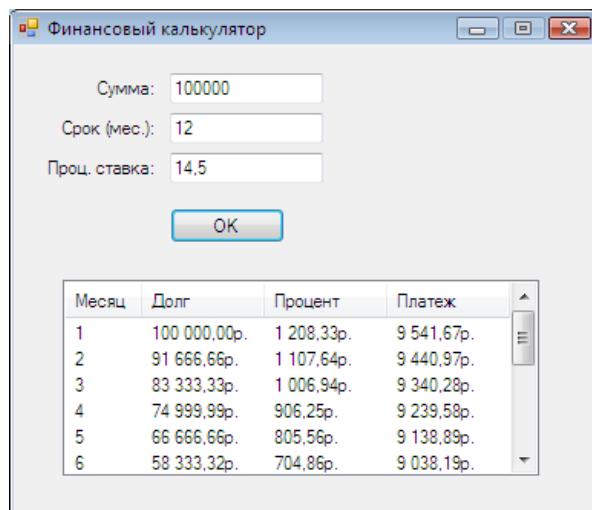


Рис. 3.23. Программа "Финансовый калькулятор"

ImageList

Компонент `ImageList` (рис. 3.24) представляет собой набор картинок. Он обычно используется другими компонентами (например, `Button` или `ToolBar`), как источник битовых образов. Компонент является невизуальным — в окне программы во время ее работы он не отображается. Во время создания формы компонент отображается в нижней части окна редактора формы.

Свойства компонента `ImageList` приведены в табл. 3.16.

Таблица 3.16. Свойства компонента `ImageList`

Свойство	Описание
<code>Images</code>	Коллекция битовых образов (объектов <code>Bitmap</code>)
<code>ImageSize</code>	Размер картинок коллекции. Уточняющее свойство <code>Width</code> определяет ширину картинки, <code>Height</code> — высоту
<code>TransparentColor</code>	Прозрачный цвет. Участки картинки, окрашенные этим цветом, не отображаются
<code>ColorDepth</code>	Глубина цвета — количество байтов, используемых для кодирования цвета точки (пикселя)

Коллекция битовых образов формируется во время разработки формы из заранее подготовленных картинок. Формат исходных картинок может быть практически любым (BMP, GIF, JPEG, PNG, ICO). Картинки должны быть одного размера и иметь одинаковый цвет фона.

Формируется коллекция картинок путем добавления в нее элементов. Сначала, после того как компонент `ImageList` будет добавлен в форму, надо задать размер картинок коллекции (присвоить значение свойству `Size`), определить прозрачный цвет (присвоить значение свойству `TransparentColor`) и задать глубину цветовой палитры (присвоить значение свойству `ColorDepth`). После этого можно приступить к формированию коллекции. Чтобы добавить в коллекцию элемент, надо: в строке свойства `Images` щелкнуть на кнопке с тремя точками; в появившемся окне **Images Collection Editor** щелкнуть на кнопке **Add**; в окне **Открыть** указать (выбрать) файл картинки. В качестве примера на рис. 3.25 приведен вид окна **Images Collection Editor**, в котором отображается содержимое коллекции картинок.

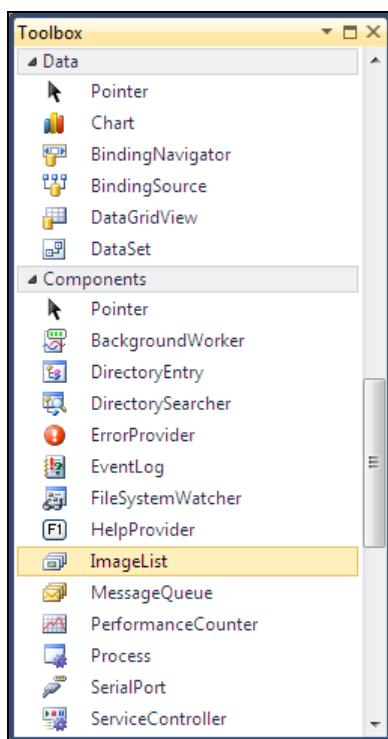


Рис. 3.24. Компонент `ImageList`

Следует обратить внимание: коллекция картинок объекта `ImageList` представляет собой совокупность битовых образов, которые находятся в файле ресурсов

(resx-файл). Эти битовые образы формируются в процессе создания коллекции путем преобразования содержимого файлов, указанных во время формирования коллекции. Таким образом, файлы иллюстраций, из которых сформирована коллекция, во время работы программы не нужны.

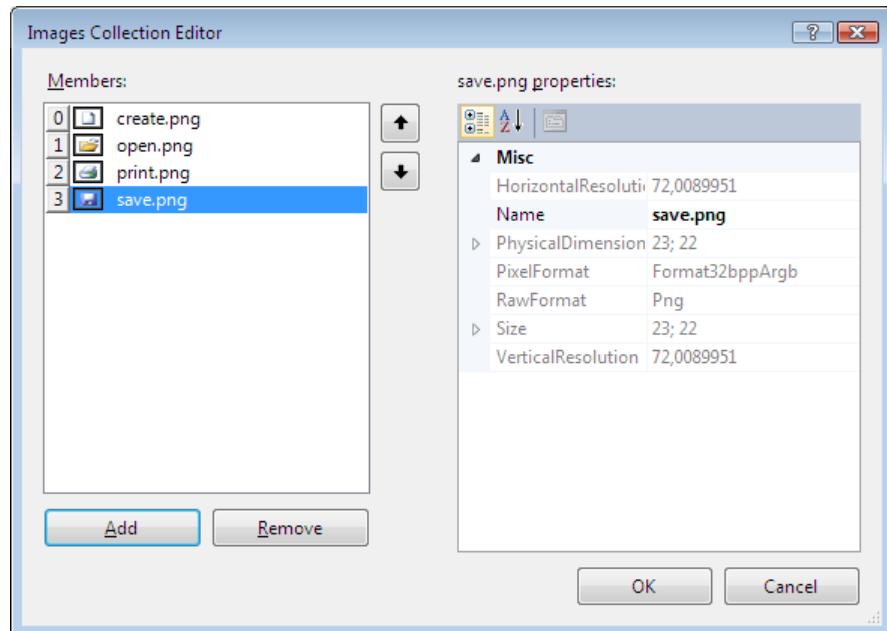


Рис. 3.25. Пример коллекции картинок

ToolTip

Компонент *ToolTip* (рис. 3.26) обеспечивает отображение подсказок при позиционировании указателя мыши на компонентах формы (рис. 3.27). Свойства компонента приведены в табл. 3.17.

Таблица 3.17. Свойства компонента *ToolTip*

Свойство	Описание
Active	Разрешает (<i>True</i>) или запрещает (<i>False</i>) отображение подсказок
AutoPopDelay	Время отображения подсказки
InitialDelay	Время, в течение которого указатель мыши должен быть неподвижным, чтобы появилась подсказка
ReshowDelay	Время задержки отображения подсказки после перемещения указателя мыши с одного компонента на другой
IsBalloon	Если значение свойства равно <i>True</i> , то окно подсказки отображается в форме облака

Таблица 3.17 (окончание)

Свойство	Описание
ToolTipTitle	Заголовок окна отображения подсказки
ToolTipIcon	Значок, отображаемый в окне подсказки

После того как компонент `ToolTip` будет добавлен в форму, у других компонентов становится доступным свойство `ToolTip` `on`, которое и определяет текст подсказки, отображаемой в результате позиционирования указателя мыши на компоненте.

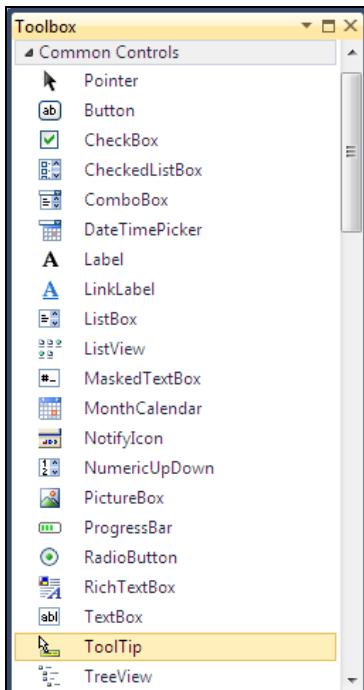
Рис. 3.26. Компонент `ToolTip`

Рис. 3.27. Пример подсказки

Panel

Компонент **Panel** (рис. 3.28) представляет собой поверхность, на которую можно поместить другие компоненты. Обычно он используется для привязки компонентов к одной из границ формы. Например, если панель привязана к нижней границе формы, то независимо от размера окна (при изменении размера окна во время работы программы) кнопки, находящиеся на панели, всегда будут находиться в нижней его части. Панель позволяет легко управлять компонентами, которые на ней находятся. Например, чтобы сделать недоступными компоненты панели, достаточно присвоить значение `False` свойству `Enabled` панели.

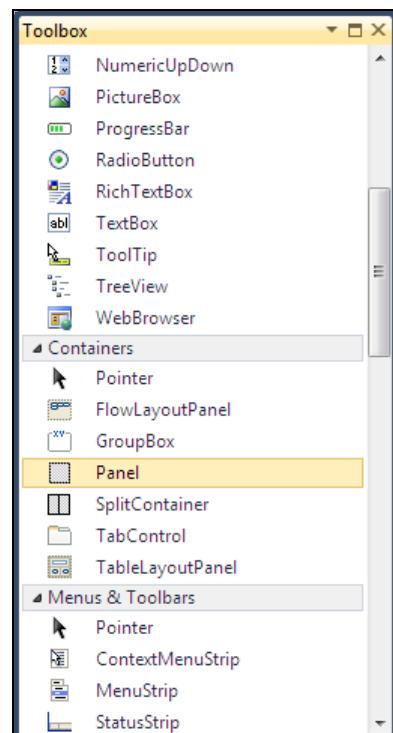


Рис. 3.28. Компонент *Panel*

Свойства компонента *Panel* приведены в табл. 3.18.

Таблица 3.18. Свойства компонента *Panel*

Свойство	Описание
Dock	Определяет границу формы, к которой привязана (прикреплена) панель. Панель может быть прикреплена к левой (<code>Left</code>), правой (<code>Right</code>), верхней (<code>Top</code>) или нижней (<code>Bottom</code>) границе
BorderStyle	Вид границы панели: <code>FixedSingle</code> — рамка; <code>Fixed3D</code> — объемная граница; <code>None</code> — граница не отображается
Enabled	Свойство позволяет сделать недоступными (<code>False</code>) все компоненты, которые находятся в панели
Visible	Позволяет скрыть (<code>False</code>) панель
AutoScroll	Признак необходимости отображать полосы прокрутки, если компоненты, которые находятся в панели, не могут быть выведены полностью

Свойство `Dock` позволяет "привязать" панель и, следовательно, находящиеся в ней компоненты к границе формы. В результате привязки панели к границе формы размер панели автоматически изменяется — ширина панели становится равной ширине формы (при привязке к верхней или нижней границе) или высота панели становится равной высоте формы (при привязке к левой или правой границе). На рис. 3.29 приведена форма программы просмотра иллюстраций. Кнопки перехода к следующей и предыдущей иллюстрациям размещены в панели, свойство `Dock` которой равно `Bottom`.

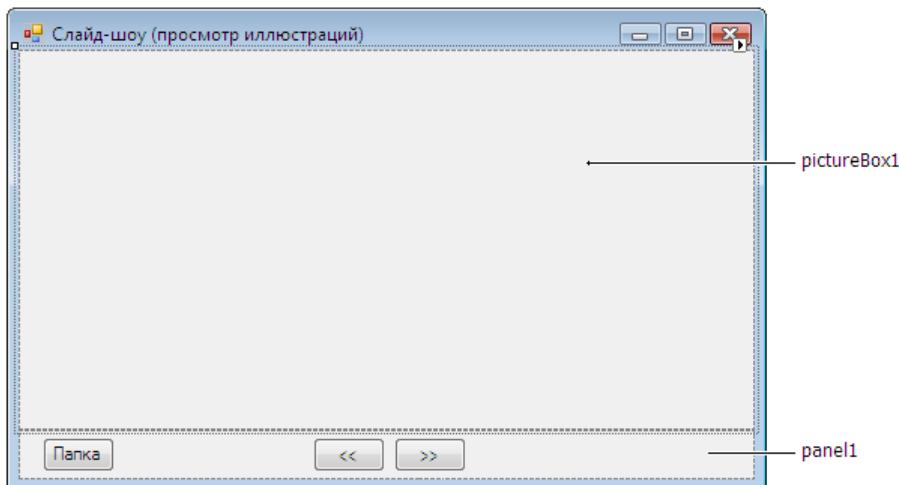


Рис. 3.29. Кнопки размещены на панели

CheckedListBox

Компонент `CheckedListBox` (рис. 3.30) представляет собой список, перед каждым элементом которого находится переключатель `CheckBox`. Свойства компонента `CheckedListBox` приведены в табл. 3.19.

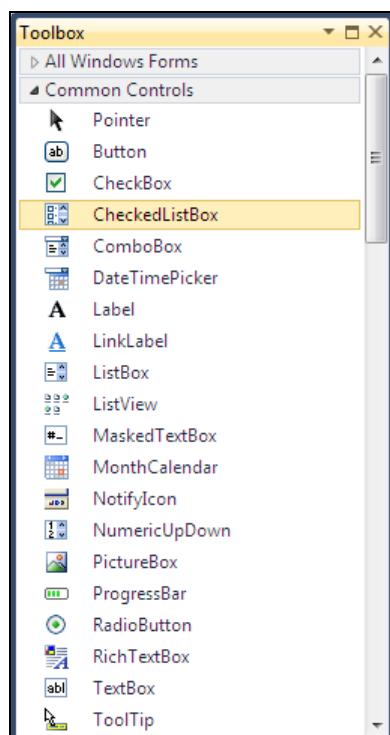
Таблица 3.19. Свойства компонента `CheckedListBox`

Свойство	Описание
<code>Items</code>	Элементы списка — коллекция строк
<code>Items.Count</code>	Количество элементов списка
<code>Sorted</code>	Признак необходимости автоматической сортировки (<code>True</code>) списка после добавления очередного элемента
<code>CheckOnClick</code>	Способ пометки элемента списка. Если значение свойства равно <code>False</code> , то первый щелчок выделяет элемент списка (строку), а второй устанавливает в выбранное состояние переключатель. Если значение свойства равно <code>True</code> , то щелчок на элементе списка выделяет элемент и устанавливает во включенное состояние переключатель

Таблица 3.19 (окончание)

Свойство	Описание
CheckedItems	Свойство CheckedItems представляет собой коллекцию, элементы которой содержат выбранные элементы списка
CheckedItems.Count	Количество выбранных элементов списка, переключатели которых установлены в выбранное состояние
CheckedIndices	Свойство CheckedIndices представляет собой коллекцию, элементы которой содержат номера выбранных (помеченных) элементов списка
ScrollAlwaysVisible	Признак необходимости всегда отображать вертикальную полосу прокрутки. Если значение свойства равно False, то полоса прокрутки отображается, только если все элементы списка нельзя отобразить при заданном размере компонента
MultiColumn	Признак необходимости отображать список в несколько колонок. Количество отображаемых колонок зависит от количества элементов и размера области отображения списка
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа DropDownList и DropDown) или с учетом (для компонента типа Simple) размера области списка или области ввода
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

Сформировать список компонента `CheckedListBox` можно во время работы программы, применив метод `Add` к свойству `Items`. В инструкции вызова метода `Add` в качестве второго параметра можно указать константу `True`. В этом случае переключатель перед элементом списка будет установлен в выбранное состояние.

Рис. 3.30. Компонент `CheckedListBox`

Timer

Компонент Timer (рис. 3.31) генерирует последовательность событий Tick. Обычно он используется для активизации с заданным периодом некоторых действий. Компонент является невизуальным (во время работы программы в окне не отображается). Свойства компонента приведены в табл. 3.20.

Таблица 3.20. Свойства компонента Timer

Свойство	Описание
Interval	Период генерации события Tick. Задается в миллисекундах
Enabled	Разрешение работы. Разрешает (значение True) или запрещает (значение False) генерацию события Tick

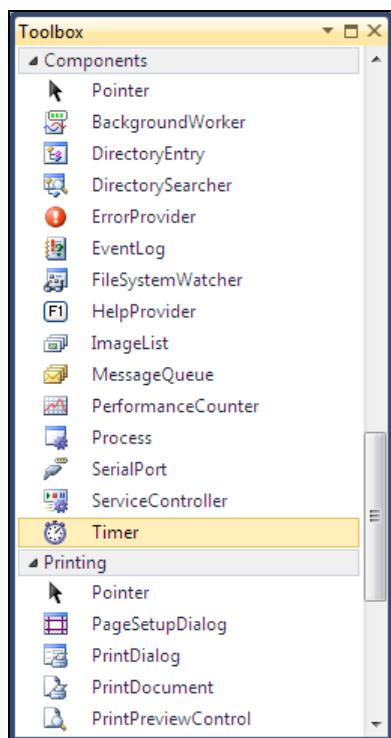


Рис. 3.31. Компонент Timer

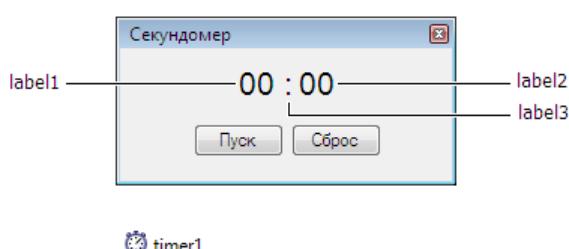


Рис. 3.32. Форма программы "Секундомер"

Программа "Секундомер" (ее форма приведена на рис. 3.32, а текст — в листинге 3.12) демонстрирует использование компонента Timer. Во время создания формы свойству Interval таймера надо присвоить значение 500. Кнопка button1 предназначена как для запуска секундомера, так и для его остановки. В начале работы программы значение свойства Enabled компонента Timer равно False, поэтому таймер не работает (не генерирует события Tick). Процедура обработки

события Click на кнопке button1 проверяет состояние таймера и, если он не работает, присваивает значение True свойству Enabled компонента Timer и тем самым запускает таймер. Процедура обработки события Tick инвертирует значение свойства Visible компонента label2 (в результате двоеточие мигает с периодом 0,5 секунд) и, если двоеточие скрыто, завершает свою работу. Если двоеточие отображается, то она увеличивает счетчик времени и выводит в поле компонентов label1 и label3 значения счетчиков минут и секунд (величину интервала с момента запуска секундомера). Если секундомер работает, то щелчок на кнопке Стоп (button1) останавливает секундомер.

Листинг 3.12. Секундомер

```
// минуты, секунды, миллисекунды
int m, s;

public Form1()
{
    InitializeComponent();

    // настройка таймера: сигнал от таймера - каждые 0.5 секунды
    timer1.Interval = 500;

    // обнуление показаний
    m = 0; s = 0;

    label1.Text = "00";
    label2.Text = "00";
    label3.Visible = true;
}

// щелчок на кнопке Пуск/Стоп
private void button1_Click(object sender, EventArgs e)
{
    if (timer1.Enabled)
    {
        // таймер работает, остановить таймер
        timer1.Enabled = false;

        // изменить текст на кнопке и сделать ее доступной
        button1.Text = "Пуск";
        button2.Enabled = true;
    }
    else
    {
        // таймер не работает, запускаем таймер
        timer1.Enabled = true;
    }
}
```

```
// изменить текст на кнопке и сделать ее недоступной
button1.Text = "Стоп";
button2.Enabled = false;
}

}

// щелчок на кнопке Сброс
private void button2_Click(object sender, EventArgs e)
{
    m = 0; s = 0;

    label1.Text = "00";
    label2.Text = "00";

    // label4.Text = ":";

}

// сигнал от таймера
private void timer1_Tick(object sender, EventArgs e)
{
    // двоеточие мигает с периодом 0.5 сек
    if (label3.Visible)
    {
        if (s < 59)
        {
            s++;
            if (s < 10)
                label2.Text = "0" + s.ToString();
            else
                label2.Text = s.ToString();
        }
        else
        {
            if (m < 59)
            {
                m++;
                if (m < 10)
                    label1.Text = "0" + m.ToString();
                else
                    label1.Text = m.ToString();
                s = 0;
                label2.Text = "00";
            }
            else
            {
                m = 0;
                label1.Text = "00";
            }
        }
    }
}
```

```

label3.Visible = false;
}
else
    label3.Visible = true;
}

```

NumericUpDown

Компонент **NumericUpDown** (рис. 3.33) предназначен для ввода числовых данных. Данные можно ввести в поле редактирования путем набора на клавиатуре или изменить уже введенные данные при помощи командных кнопок **Увеличить** и **Уменьшить**, которые находятся справа от поля редактирования.

Свойства компонента **NumericUpDown** приведены в табл. 3.21.

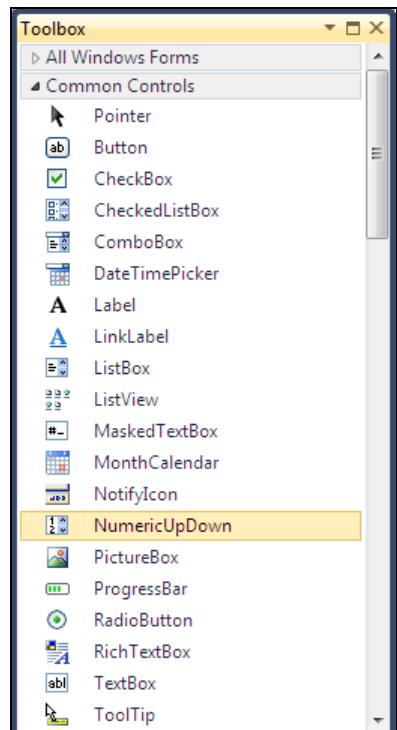


Рис. 3.33. Компонент **NumericUpDown**

Таблица 3.21. Свойства компонента **NumericUpDown**

Свойство	Описание
Value	Значение, соответствующее содержимому поля редактирования
Maximum	Максимально возможное значение, которое можно ввести в поле компонента
Minimum	Минимально возможное значение, которое можно ввести в поле компонента
Increment	Величина, на которую увеличивается или уменьшается значение свойства Value при каждом щелчке мышью на кнопках Увеличить или Уменьшить
DecimalPlaces	Количество цифр дробной части числа

Если в поле редактирования компонента `NumericUpDown` введено дробное число, то в результате нажатия клавиши <Enter> оно будет округлено до `DecimalPlaces` цифр дробной части. При этом если значение свойства `DecimalPlaces` равно нулю, то значение в поле компонента будет округлено до целого.

Программа "Таймер" (ее форма приведена на рис. 3.34, а текст — в листинге 3.13) демонстрирует использование компонента `NumericUpDown`.

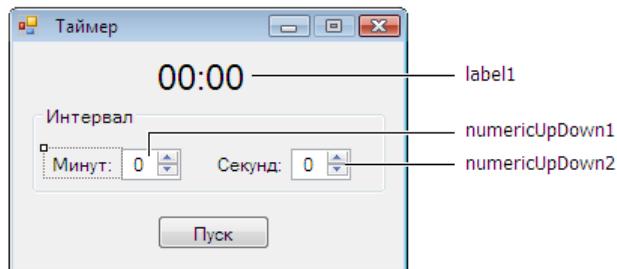


Рис. 3.34. Форма программы "Таймер"

Компоненты `numericUpDown1` и `numericUpDown2` используются для ввода значения интервала времени. Во время создания формы свойству `Maximum` обоих компонентов надо присвоить значение 59.

Листинг 3.13. Таймер

```

private DateTime t1; // время запуска таймера
private DateTime t2; // время срабатывания таймера

public Form1()
{
    InitializeComponent();

    // Настройка компонентов numericUpDown
    numericUpDown1.Maximum = 59;
    numericUpDown1.Minimum = 0;
    // Чтобы при появлении окна курсор не мигал в поле редактирования
    numericUpDown1.TabStop = false;

    numericUpDown2.Maximum = 59;
    numericUpDown2.Minimum = 0;
    numericUpDown2.TabStop = false;

    // Кнопка Пуск/Стоп недоступна
    button1.Enabled = false;
}

```

```
// Обрабатывает событие ValueChanged от компонентов
// numericUpDown1 и numericUpDown1
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    if ((numericUpDown1.Value == 0) && (numericUpDown2.Value == 0))
        button1.Enabled = false;
    else
        button1.Enabled = true;
}

// щелчок на кнопке Пуск/Стоп
private void button1_Click(object sender, EventArgs e)
{
    if (!timer1.Enabled)
    {
        // таймер не работает

        // t1 - текущее время
        // t2 = t1 + интервал
        t1 = new DateTime(DateTime.Now.Year,
                          DateTime.Now.Month, DateTime.Now.Day);
        t2 = t1.AddMinutes((double)numericUpDown1.Value);
        t2 = t2.AddSeconds((double)numericUpDown2.Value);

        groupBox1.Enabled = false;
        button1.Text = "Стоп";

        if (t2.Minute < 9)
            label1.Text = "0" + t2.Minute.ToString() + ":";
        else
            label1.Text = t2.Minute.ToString() + ":";

        if (t2.Second < 9)
            label1.Text += "0" + t2.Second.ToString();
        else
            label1.Text += t2.Second.ToString();

        // сигнал от таймера поступает каждую секунду
        timer1.Interval = 1000;

        // пуск таймера
        timer1.Enabled = true;
    }
    else
    {
        // таймер работает, останавливаем
        timer1.Enabled = false;
    }
}
```

```
button1.Text = "Пуск";
groupBox1.Enabled = true;
numericUpDown1.Value = t2.Minute;
numericUpDown2.Value = t2.Second;
}

}

// сигнал от таймера
private void timer1_Tick(object sender, EventArgs e)
{
    t2 = t2.AddSeconds(-1);

    if (t2.Minute < 9)
        label1.Text = "0" + t2.Minute.ToString() + ":";
    else
        label1.Text = t2.Minute.ToString() + ":";

    if (t2.Second < 9)
        label1.Text += "0" + t2.Second.ToString();
    else
        label1.Text += t2.Second.ToString();

    if (Equals(t1, t2))
    {
        timer1.Enabled = false;
        MessageBox.Show("Заданный интервал времени истек", "Таймер",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
    }
}
```

При изменении значения, находящегося в поле редактирования компонента `NumericUpDown`, возникает событие `ValueChanged`. Функция обработки этого события (одна для обоих компонентов) управляет доступностью кнопки **Пуск**. Она проверяет значения свойств `Value` и, если интервал не задан (равен нулю), делает кнопку недоступной. Процедура обработки события `Click`, возникающего при щелчке на кнопке `button1`, запускает или, если таймер работает, останавливает таймер. При запуске таймера в переменную `t1` записывается текущее время, а в переменную `t2` — значение, сдвинутое вперед на величину интервала (`t2 = момент запуска + интервал`). Процедура обработки события `Tick` таймера уменьшает на единицу значение `t2` и, если оно становится равным `t1`, останавливает таймер. Она же обеспечивает

чиает индикацию — выводит в поле компонента `label1`, сколько времени (минут и секунд) осталось до момента "срабатывания" таймера.

StatusStrip

Компонент `StatusStrip` (рис. 3.35) представляет собой область отображения служебной информации (часто ее называют *строкой состояния*). Страна вывода служебной информации отображается в нижней части окна программы и обычно разделена на области. В области может отображаться, например, текстовая информация или индикатор процесса (`ProgressBar`) или командные кнопки.

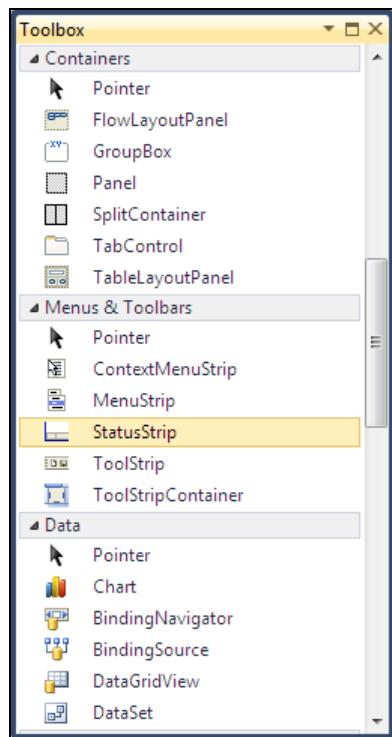


Рис. 3.35. Компонент `StatusStrip`

Чтобы в строке состояния отображался текст, в нее надо добавить элемент `StatusLabel`. Для этого нужно сделать щелчок на значке раскрывающегося списка, который отображается в поле компонента `StatusStrip`, и выбрать `StatusLabel` (рис. 3.36). После этого надо выполнить настройку компонента `toolStripStatusLabel` — задать значения свойств (табл. 3.22).

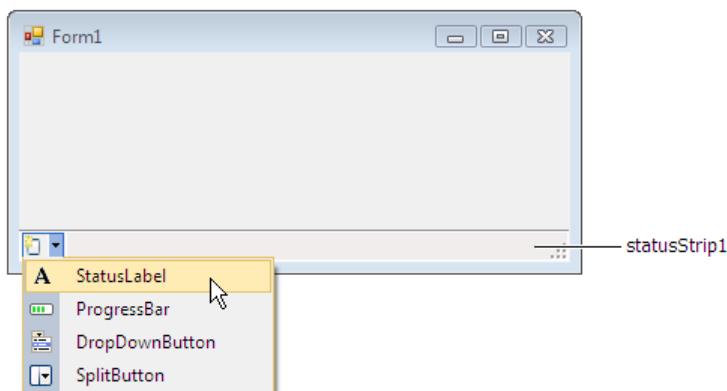
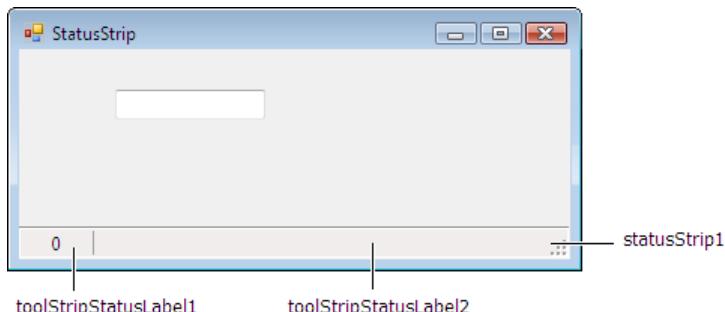


Рис. 3.36. Добавление элемента в строку состояния

Таблица 3.22. Свойства объекта *StatusLabel*

Свойство	Описание
Text	Текст, отображаемый в панели
AutoSize	Признак автоматического изменения размера панели. Если значение свойства равно <i>True</i> , то размер ширины панели зависит от ее содержания (длины текста). Если значение свойства равно <i>False</i> , то ширину панели определяет свойство <i>Size.Width</i>
BorderStyle	Вид границы панели
BorderSides	Определяет отображаемые границы
Spring	Определяет, должен (<i>True</i>) или нет (<i>False</i>) компонент занимать всю доступную область строки состояния

Следующая программа (ее форма приведена на рис. 3.37, а текст — в листинге 3.14) демонстрирует использование компонента *StatusStrip*. В строке состояния отражаются текущая дата и количество символов, которое пользователь ввел в поле редактирования. Значения свойств компонентов *toolStripStatusLabel1* приведены в табл. 3.23. Вывод информации о количестве введенных символов выполняет процедура обработки события *TextChanged* компонента *TextBox*. Отображение даты обеспечивает конструктор формы.

**Рис. 3.37.** Форма программы *StatusStrip***Таблица 3.23.** Значения свойств компонентов *toolStripStatusLabel*

Компонент	Свойство	Значение
<i>toolStripStatusLabel1</i>	AutoSize	<i>False</i>
	<i>Size.Width</i>	50
	Text	0
	BorderSides	Right
<i>toolStripStatusLabel2</i>	Spring	<i>True</i>
	TextAlign	MiddleRight

Листинг 3.14. Компонент StatusStrip

```

// конструктор формы
public Form1()
{
    InitializeComponent();

    // в поле toolStripStatusLabel2 показать текущую дату
    statusStrip1.Items[1].Text = DateTime.Now.ToString("D");
}

// изменилось содержимое поля редактирования
private void textBox1_TextChanged(object sender, EventArgs e)
{
    int len;

    len = textBox1.Text.Length; // кол-во символов в поле редактирования

    // в поле toolStripStatusLabel2 показать
    // кол-во символов, введенных в поле редактирования
    statusStrip1.Items[0].Text = len.ToString("D");
}

```

NotifyIcon

Компонент NotifyIcon (рис. 3.38) представляет собой "извещающий" значок (от англ. *notify* — извещать, напоминать), который изображает в панели задач программу, работающую в фоновом режиме. Обычно при позиционировании указателя мыши на таком значке появляется подсказка, а в результате щелчка правой кнопкой — контекстное меню, команды которого позволяют получить доступ к программе (открыть окно) или завершить ее работу.

Свойства компонента приведены в табл. 3.24.

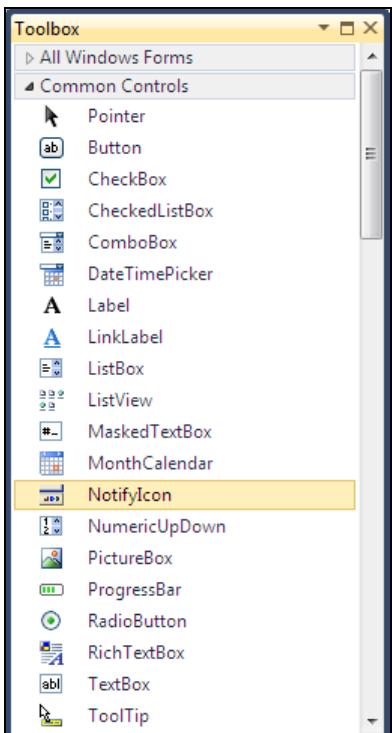
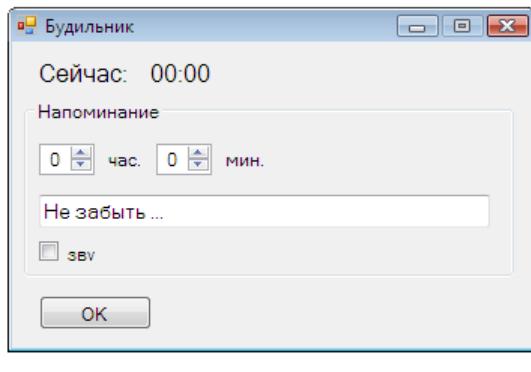


Рис. 3.38. Компонент NotifyIcon

Таблица 3.24. Свойства компонента NotifyIcon

Свойство	Описание
Icon	Значок, который отображается в панели задач
Text	Подсказка (обычно название программы). Отображается рядом с указателем мыши при позиционировании указателя на значке
ContextMenu	Ссылка на компонент ContextMenu, который обеспечивает отображение контекстного меню как результат щелчка правой кнопкой мыши на значке
Visible	Свойство позволяет скрыть (False) значок с панели задач или сделать его видимым (True)

Программа "Будильник" (ее главная форма приведена на рис. 3.39, а значения свойств компонентов — в табл. 3.25) демонстрирует использование компонента NotifyIcon.



⌚ timer1 ☰ contextMenuStrip1 ⚡ notifyIcon1

Рис. 3.39. Главная форма программы "Будильник"**Таблица 3.25.** Значения свойств компонентов

Компонент	Свойство	Значение
timer1	Interval	1000
	Enabled	False
contextMenuStrip1	toolStripMenuItem1.Text	Показать
	toolStripMenuItem2.Text	О программе
	toolStripMenuItem3.Text	Завершить
notifyIcon1	Icon	🔔
	Visible	False
	ContextMenu	contextMenuStrip1

В окне программы "Будильник" после ее запуска отображается текущее время. После того как пользователь установит время сигнала, задаст текст сообщения и сделает щелчок на кнопке **OK**, окно программы закроется и в системной области панели задач появится колокольчик — значок, изображающийирующую программу "Будильник". При позиционировании указателя мыши на колокольчике появляется подсказка, в которой отображается время, когда будильник должен подать сигнал и текст напоминания. В контекстном меню программы, которое появляется в результате щелчка правой кнопкой мыши, три команды: **Показать**, **О программе** и **Завершить** (рис. 3.40).

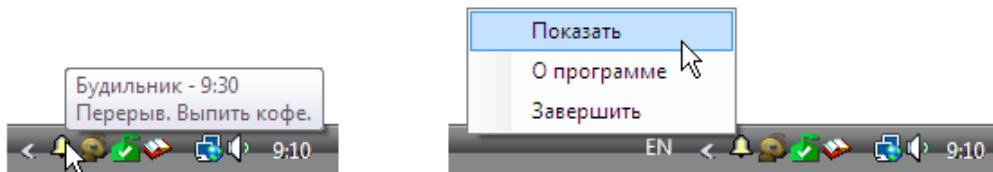


Рис. 3.40. Значок программы "Будильник" в панели задач: подсказка и контекстное меню

В заданный момент времени на экране появляется окно сообщения. В данной программе окно сообщения это обычное окно, такое же, как и главное окно программы. Чтобы его создать, в проект надо добавить форму — в меню **Project** выбрать команду **Add Windows Forms**. После этого на форму Form2 надо добавить два компонента **Label** и командную кнопку (рис. 3.41). Следует обратить внимание на то, что для кнопки **OK** в программе нет процедуры обработки события **Click**, тем не менее, в результате щелчка на кнопке окно сообщения закрывается. Это происходит потому, что свойству **DialogResult** кнопки **OK** присвоено значение **OK**.

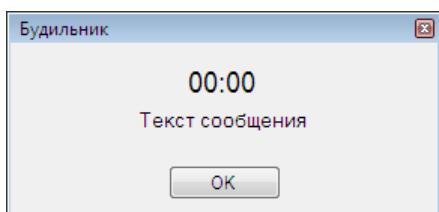


Рис. 3.41. Форма сообщения

Объявление класса главной формы приведено в листинге 3.15. Обратите внимание, в объявление класса добавлена ссылка на библиотеку **winmm.dll**, в которой находится функция **PlaySound**, обеспечивающая воспроизведение звукового сигнала. Объявление класса формы сообщения приведено в листинге 3.16.

Листинг 3.15. Будильник (объявление класса главной формы)

```
public partial class Form1 : Form
{
    // Функция PlaySound, обеспечивающая воспроизведение
    // wav-файлов, находится в библиотеке winmm.dll.
```

```
// Подключим эту библиотеку
[System.Runtime.InteropServices.DllImport("winmm.dll")]
private static extern
    Boolean PlaySound(string lpszName, int hModule, int dwFlags);

private DateTime alarm; // время сигнала
private bool isSet; // true - будильник установлен (работает)

public Form1()
{
    InitializeComponent();

    // параметры компонентов numericUpDown
    numericUpDown1.Maximum = 23;
    numericUpDown1.Minimum = 0;

    numericUpDown2.Maximum = 59;
    numericUpDown2.Minimum = 0;

    numericUpDown1.Value = DateTime.Now.Hour;
    numericUpDown2.Value = DateTime.Now.Minute;

    notifyIcon1.Visible = false;

    // период обработки сигнала от таймера
    timer1.Interval = 1000;
    timer1.Enabled = true;

    label2.Text = DateTime.Now.ToString();
}

// щелчок на кнопке OK
private void button1_Click(object sender, EventArgs e)
{
    // установить время сигнала
    alarm = new DateTime(DateTime.Now.Year,
        DateTime.Now.Month,
        DateTime.Now.Day,
        Convert.ToInt16(numericUpDown1.Value),
        Convert.ToInt16(numericUpDown2.Value),
        0, 0);

    // Если установленное время будильника меньше текущего,
    // нужно увеличить дату срабатывания на единицу (+1 день)
    if (DateTime.Compare(DateTime.Now, alarm) > 0)
        alarm = alarm.AddDays(1);
}
```

```
// Настройка подсказки, которая будет отображаться
// при позиционировании указателя мыши на значке
// программы, появляющемся в панели задач
notifyIcon1.Text = "Будильник - " + alarm.ToShortTimeString() +
    "\n" + textBox1.Text;

isSet = true;

// скрыть главное окно
this.Hide();

// показать значок в панели задач
notifyIcon1.Visible = true;
}

// сигнал от таймера
private void timer1_Tick(object sender, EventArgs e)
{
    label2.Text = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

    // будильник установлен?
    if (isSet)
    {
        // время срабатывания будильника?
        if (DateTime.Compare(DateTime.Now, alarm) > 0)
        {
            // текущее время совпало с установленным
            isSet = false;

            if (checkBox1.Checked)
            {
                // звук
                PlaySound(Application.StartupPath + "\\ring.wav", 0, 1);
            }
        }

        Form2 frm; // окно сообщения

        // см. конструктор формы Form2
        frm = new Form2(DateTime.Now.ToShortTimeString(),
                         this.textBox1.Text);

        frm.ShowDialog(); // показать окно сообщения
                          // как модальный диалог

        // Здесь пользователь закрыл окно сообщения.
        // Показать главное окно
        this.Show();
    }
}
```

```
// выбор в контекстном меню команды Показать/Свернуть
private void toolStripMenuItem1_Click(object sender, EventArgs e)
{
    if (this.Visible)
    {
        this.Hide();
    }

    else
    {
        this.Show();
        notifyIcon1.Visible = false;
    }
}

// о программе
private void toolStripMenuItem2_Click(object sender, EventArgs e)
{
}

// выбор в контекстном меню команды Завершить
private void toolStripMenuItem3_Click(object sender, EventArgs e)
{
    this.Close();
}

private void notifyIcon1_Click(object sender, EventArgs e)
{
    // если нет контекстного меню, то открыть окно программы можно так:
    /*
        setIsSet = false;
        this.Show();
        notifyIcon1.Visible = false;
    */
}
}
```

Листинг 3.16. Будильник (объявление класса формы сообщения)

```
public partial class Form2 : Form
{
    public Form2()
    {
        InitializeComponent();
    }
}
```

```

/*
Так как нам нужно, чтобы при появлении формы в полях label
отображалось текущее время и сообщение, создадим конструктор
с параметрами, который будет устанавливать значение свойства Text
этих компонентов. Форму будем создавать, используя этот,
а не стандартный конструктор.
*/

```

```

// мой конструктор
public Form2(String s1, String s2)
{
    InitializeComponent();
    //
    label1.Text = s1;
    label2.Text = s2;
}

```

ToolStrip

Компонент ToolStrip (рис. 3.42) — полоса (панель) инструментов используется для размещения в нем других компонентов. В панель инструментов можно поместить командную кнопку, поле отображения текста, поле редактирования, раскрывающийся список.

Свойства компонента ToolStrip приведены в табл. 3.26.

Для того чтобы в панель инструментов добавить, например, командную кнопку, надо выбрать компонент ToolStrip, сделать щелчок на значке раскрывающегося списка, который отображается в его поле, и выбрать Button (рис. 3.43). После этого надо выполнить настройку добавленной кнопки toolStripButton — задать значения свойств (табл. 3.26).

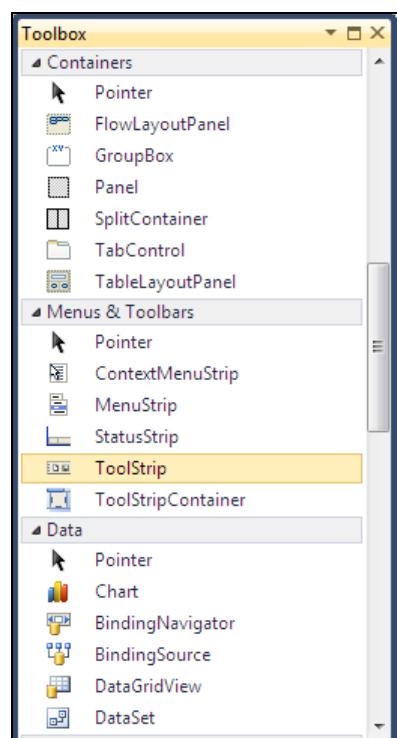


Рис. 3.42. Компонент ToolStrip

Таблица 3.26. Свойства компонента ToolStrip

Свойство	Описание
Buttons	Items — коллекция компонентов, находящихся в панели инструментов
Dock	Граница родительского компонента (формы), к которой "привязана" панель инструментов: к верхней (Top), левой (Left), нижней (Bottom) или правой (Right)
Visible	Свойство позволяет скрыть (False) или сделать видимой (True) панель инструментов
Enabled	Свойство управляет доступом к компонентам панели инструментов. Если значение свойства равно False, то все компоненты недоступны

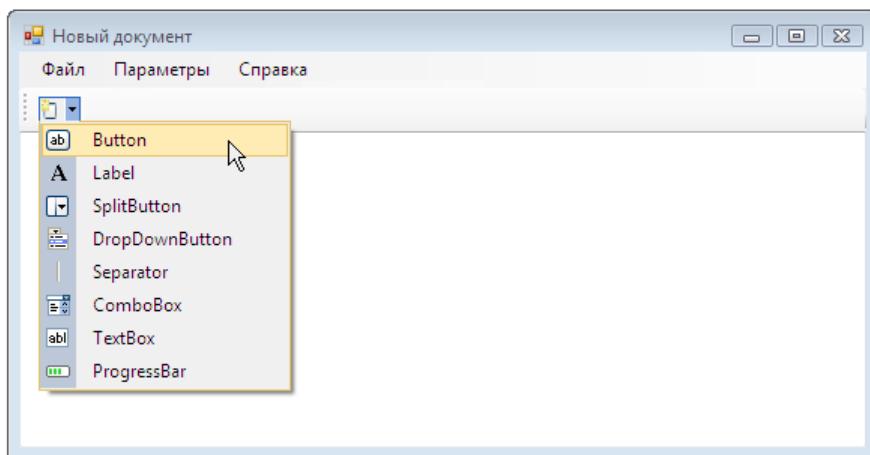


Рис. 3.43. Добавление элемента в панель инструментов

В качестве примера на рис. 3.44 приведена форма программы MEdit. В панели инструментов находятся четыре кнопки и разделитель (Separator). Значения свойств компонентов, находящихся в панели инструментов, приведены в табл. 3.27. Следует обратить внимание, фон битовых образов для командных кнопок — пурпурный (magenta). Именно этот цвет и указан в качестве значения свойства ImageTransparentColor.

Таблица 3.27. Значения свойств компонентов

Компонент	Свойство	Значение
toolStripButton1	DisplayStyle	Image
	Image	
	ImageTransparentColor	Magenta
	ToolTipText	Новый документ

Таблица 3.27 (окончание)

Компонент	Свойство	Значение
toolStripButton2	DisplayStyle	Image
	Image	
	ImageTransparentColor	Magenta
	ToolTipText	Открыть
toolStripButton3	DisplayStyle	Image
	Image	
	ImageTransparentColor	Magenta
	ToolTipText	Сохранить
toolStripButton4	DisplayStyle	Image
	Image	
	ImageTransparentColor	Magenta
	ToolTipText	Печать

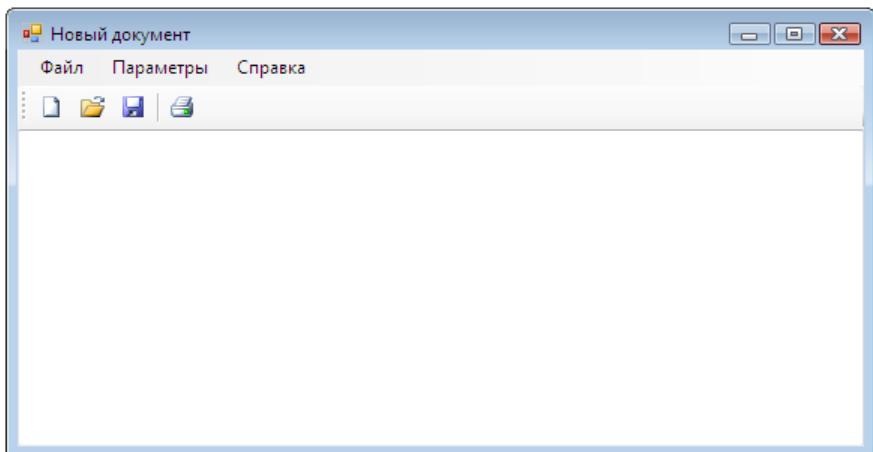


Рис. 3.44. Форма программы MEdit

После того как панель инструментов будет создана, можно приступить к созданию процедур обработки событий.

MenuStrip

Компонент *MenuStrip* (рис. 3.45) представляет собой главное меню программы.

После того как в форму будет добавлен компонент *MenuStrip*, в верхней части формы появляется строка меню, в начале которой находится область ввода текста (прямоугольник, внутри которого находится текст "Type Here" — "Печатай здесь").

Чтобы создать первый пункт меню, надо сделать щелчок в области ввода текста и ввести название пункта меню, например, **Файл**. Как только будет введена первая буква, справа и снизу от области ввода появятся еще две области ввода текста. В нижнюю область надо ввести название команды меню, например, **Новый**, а в правую — название следующего пункта меню, например, **Параметры**. В качестве примера на рис. 3.46 приведена форма программы MEdit во время формирования структуры меню.

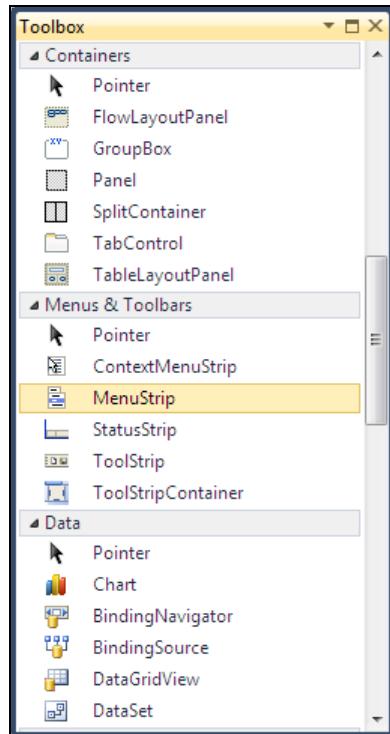


Рис. 3.45. Компонент ToolStrip

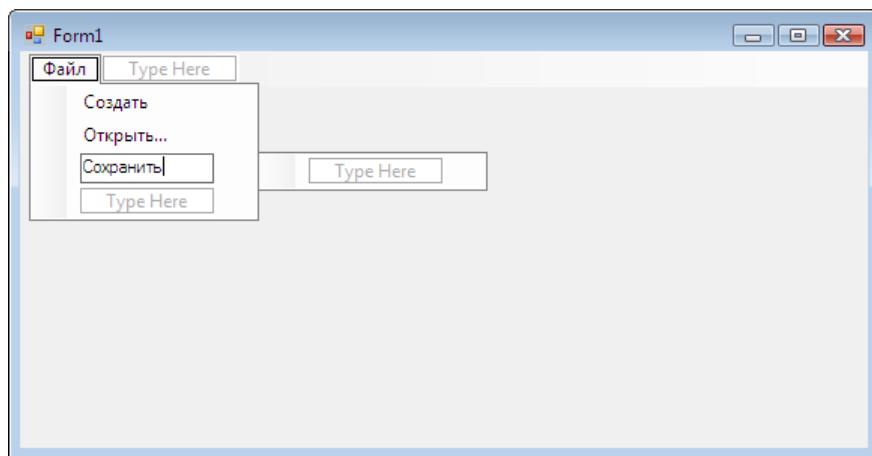


Рис. 3.46. Процесс формирования структуры меню

Кроме команды в меню можно добавить разделитель — горизонтальную линию. Разделитель обычно служит для объединения в группы однотипных команд. Чтобы добавить его в список команд, следует выделить команду, перед которой надо поместить разделитель, и из контекстного меню выбрать команду **Separator** (рис. 3.47).

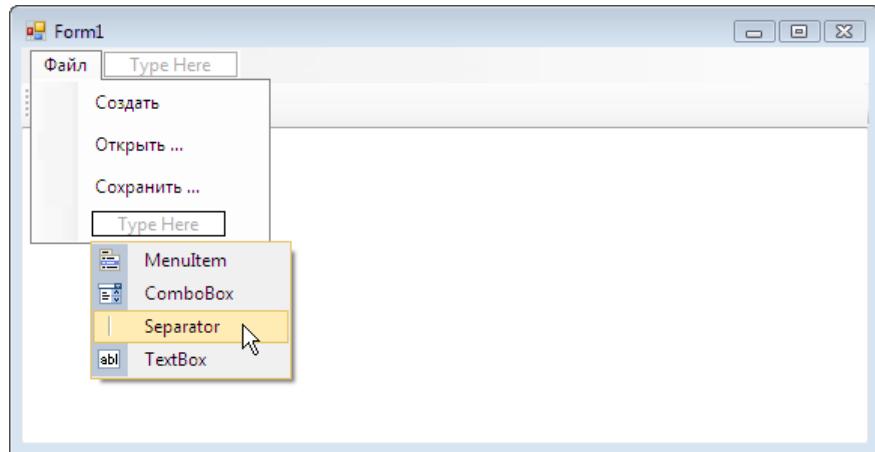


Рис. 3.47. Вставка в список команд разделителя

После того как структура меню будет сформирована, можно выполнить настройку меню. Настройка выполняется путем изменения значений свойств пунктов меню (объектов типа ToolStripMenuItem). Свойства объекта ToolStripMenuItem приведены в табл. 3.28.

Таблица 3.28. Свойства объекта ToolStripMenuItem

Свойство	Описание
Text	Название элемента меню
Image	Картинка, которая отображается рядом с командой
Enabled	Признак доступности элемента меню. Если значение свойства равно False, то элемент меню недоступен (в результате щелчка на элементе меню событие Click не происходит, название элемента меню отображается инверсным, по отношению к доступному пункту меню, цветом)
Checked	Признак того, что элемент меню выбран. Если значение свойства равно True, то элемент помечается галочкой. Свойство Checked обычно применяется для тех элементов меню, которые используются для отображения параметров, например, пункт меню Отображать строку состояния
ShortcutKeys	Свойство определяет комбинацию клавиш, нажатие которой активизирует выполнение команды

Объект ToolStripMenuItem способен воспринимать событие Click, которое происходит в результате щелчка на элементе меню, нажатия клавиши <Enter>, если элемент меню выбран, или в результате нажатия функциональной клавиши, указанной в свойстве ShortcutKeys.

В качестве примера в табл. 3.29 приведены свойства компонентов главного меню программы (редактора текста) MEdit.

Таблица 3.29. Значение свойств элементов меню программы MEdit

Свойство	Значение
fileToolStripMenuItem.Text	Файл
toolStripMenuItem1.Text	Создать
toolStripMenuItem1.Image	
toolStripMenuItem2.Text	Открыть
toolStripMenuItem2.Image	
toolStripMenuItem3.Text	Сохранить
toolStripMenuItem3.Image	
toolStripMenuItem4.Text	Печать
toolStripMenuItem4.Image	
toolStripMenuItem5.Text	Выход
toolStripMenuItem5.ShortcutKeys	Alt+F4
paramToolStripMenuItem.Text	Параметры
toolStripMenuItem6.Text	Панель инструментов
toolStripMenuItem6.Checked	True
toolStripMenuItem7.Text	Шрифт
helpToolStripMenuItem.Text	Справка
toolStripMenuItem8.Text	Справочная информация
toolStripMenuItem8.Text	О программе

OpenFileDialog

Компонент OpenFileDialog (рис. 3.48) представляет собой диалоговое окно (диалог) **Открыть**. Свойства компонента приведены в табл. 3.30.

Таблица 3.30. Свойства компонента OpenFileDialog

Свойство	Описание
Title	Текст в заголовке окна. Если значение свойства не задано, то в заголовке отображается текст Открыть
InitialDirectory	Каталог, содержимое которого отображается при появлении диалога на экране
Filter	Свойство задает один или несколько фильтров файлов. В окне отображаются только те файлы, имена которых соответствуют выбранному фильтру. Фильтр задается строкой вида Описание Маска . Например, фильтр Текст*.txt задает, что в окне диалога следует отображать только текстовые файлы. Фильтр может состоять из нескольких элементов, например: Текст*.txt Все файлы!*

Таблица 3.30 (окончание)

Свойство	Описание
FilterIndex	Если фильтр состоит из нескольких элементов, например, Текст *.txt Все файлы *.* , то значение свойства задает фильтр, который будет использоваться в момент появления диалога на экране
FileName	Имя выбранного пользователем файла
RestoreDirectory	Признак необходимости отображать содержимое каталога, указанного в свойстве <code>InitialDirectory</code> , при каждом появлении окна. Если значение свойства равно <code>False</code> , то при следующем появлении окна отображается содержимое каталога, выбранного пользователем в предыдущий раз

Отображение диалога **Открыть** обеспечивает метод `ShowDialog`. Его значением является код клавиши, нажатием которой пользователь завершил диалог (закрыл окно).

В качестве примера в листинге 3.17 приведен фрагмент программы MEdit — функция обработки события `Click` на команде **Открыть** меню **Файл**.

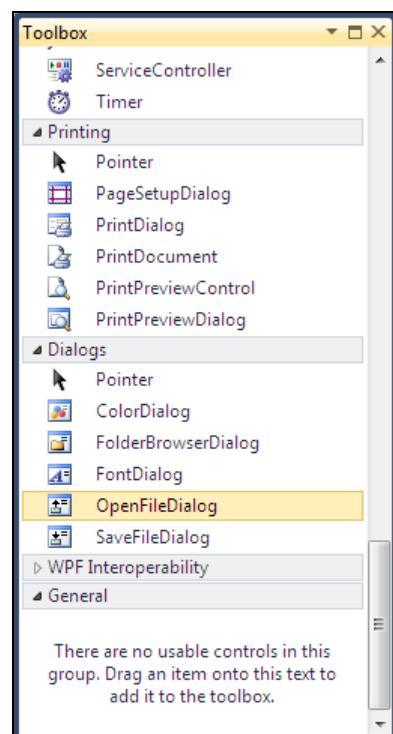


Рис. 3.48. Компонент OpenFileDialog

Листинг 3.17. Использование диалога OpenFileDialog

```
// выбор в меню Файл команды Открыть
private void FileOpenToolStripMenuItem_Click(object sender, EventArgs e)
{
    openFileDialog1.FileName = string.Empty;

    // отобразить диалог Открыть
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        fn = openFileDialog1.FileName;
    }
}
```

```
// отобразить имя файла в заголовке окна
this.Text = fn;

try
{
    // считываем данные из файла
    System.IO.StreamReader sr = new System.IO.StreamReader(fn);

    textBox1.Text = sr.ReadToEnd();
    textBox1.SelectionStart = textBox1.TextLength;

    sr.Close();
}
catch (Exception exc)
{
    MessageBox.Show("Ошибка чтения файла.\n" +
        exc.ToString(), "MEdit",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}

}
```

SaveFileDialog

Компонент `SaveFileDialog` (рис. 3.49) представляет собой стандартное диалоговое окно **Сохранить**. Свойства компонента приведены в табл. 3.31.

Таблица 3.31. Свойства компонента SaveFileDialog

Свойство	Описание
Title	Текст в заголовке окна. Если значение свойства не задано, то в заголовке отображается текст Сохранить как
FileName	Полное имя файла, которое задал пользователь. В общем случае оно образуется из имени каталога, содержимое которого отображается в диалоговом окне, имени файла, которое пользователь ввел в поле Имя файла , и расширения, заданного значением свойства DefaultExt
DefaultExt	Расширение файла по умолчанию. Если пользователь в поле Имя файла не укажет расширение, то к имени файла будет добавлено расширение (при условии, что значение свойства AddExtension равно True), заданное значением этого свойства
InitialDirectory	Каталог, содержимое которого отображается при появлении диалога на экране
RestoreDirectory	Признак необходимости отображать содержимое каталога, указанного в свойстве InitialDirectory, при каждом появлении окна. Если значение свойства равно False, то при следующем появлении окна отображается содержимое каталога, выбранного пользователем в предыдущий раз

Таблица 3.31 (окончание)

Свойство	Описание
CheckPathExists	Признак необходимости проверки существования каталога, в котором следует сохранить файл. Если указанного каталога нет, то выводится информационное сообщение
CheckFileExists	Признак необходимости проверки существования файла с заданным именем. Если значение свойства равно True и файл с указанным именем уже существует, появляется окно запроса, в котором пользователь может подтвердить необходимость замены (перезаписи) существующего файла
Filter	Свойство задает один или несколько фильтров файлов. В окне отображаются только те файлы, имена которых соответствуют выбранному фильтру. Фильтр задается строкой вида Описание Маска. Например, фильтр Текст *.txt задает, что в окне диалога следует отображать только текстовые файлы. Фильтр может состоять из нескольких элементов, например: Текст *.txt Все файлы *.*
FilterIndex	Если фильтр состоит из нескольких элементов, например, Текст *.txt Все файлы *.* , то значение свойства задает фильтр, который будет использоваться в момент появления диалога на экране

Отображение диалога **Сохранить** обеспечивает метод ShowDialog. Его значением является код клавиши, нажатием которой пользователь завершил диалог (закрыл окно).

В качестве примера в листинге 3.18 приведен фрагмент программы MEdit — функция SaveDocument, которая записывает в файл текст, находящийся в поле редактирования.

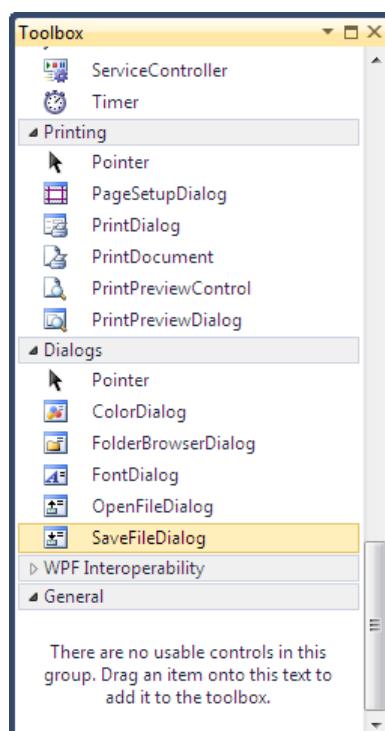


Рис. 3.49. Компонент SaveFileDialog

Листинг 3.18. Использование диалога SaveFileDialog

```
// Проверяет, есть ли изменения в тексте, и, если изменения есть,
// сохраняет текст в файле.
// Функция возвращает -1, если пользователь отказался от выполнения
// операции (нажал в окне Сохранить кнопку Отмена).
// Сохранить документ
private int SaveDocument()
{
    int result = 0;

    if (fn == string.Empty)
    {
        // отобразить диалог Сохранить
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            // отобразить имя файла в заголовке окна
            fn = saveFileDialog1.FileName;
            this.Text = fn;
        }
        else result = -1;
    }

    // сохранить файл
    if (fn != string.Empty)
    {
        try
        {
            // получим информацию о файле fn
            System.IO.FileInfo fi = new System.IO.FileInfo(fn);

            // поток для записи (перезаписываем файл)
            System.IO.StreamWriter sw = fi.CreateText();

            // записываем данные
            sw.Write(textBox1.Text);

            // закрываем поток
            sw.Close();
            result = 0;
        }
        catch (Exception exc)
        {
            MessageBox.Show(exc.ToString(), "NkEdit",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
    return result;
}
```

Полный текст программы MEdit (объявление класса главной формы, демонстрирующий использование компонентов ToolStrip, ToolTip, OpenFileDialog, SaveFileDialog, PrintDialog и FontDialog, приведен в листинге 3.19).

Листинг 3.19. Объявление класса главной формы программы MEdit

```
public partial class Form1 : Form
{
    // имя файла
    private string fn = string.Empty;

    private bool docChanged = false; // true - в тексте есть изменения

    public Form1()
    {
        InitializeComponent();

        textBox1.ScrollBars = ScrollBars.Vertical;
        textBox1.Text = string.Empty;

        this.Text = "NkEdit - Новый документ";

        // отобразить панель инструментов
        toolStrip1.Visible = true;
        ParamToolStripMenuItem.Checked = true;

        // назначаем файл справки
        // helpProvider1.HelpNamespace = "m_edit.chm";

        // настройка компонента openFileDialog1
        openFileDialog1.DefaultExt = "txt";
        openFileDialog1.Filter = "текст (*.txt)";
        openFileDialog1.Title = "Открыть документ";
        openFileDialog1.Multiselect = false;

        // настройка компонента saveDialog1
        saveFileDialog1.DefaultExt = "txt";
        saveFileDialog1.Filter = "текст (*.txt)";
        saveFileDialog1.Title = "Сохранить документ";
    }

    // открывает документ
    private void OpenDocument()
    {
        openFileDialog1.FileName = string.Empty;
```

```
// отобразить диалог Открыть
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    fn = openFileDialog1.FileName;

    // отобразить имя файла в заголовке окна
    this.Text = fn;

    try
    {
        // считываем данные из файла
        System.IO.StreamReader sr = new System.IO.StreamReader(fn);

        textBox1.Text = sr.ReadToEnd();
        textBox1.SelectionStart = textBox1.TextLength;

        sr.Close();
    }
    catch (Exception exc)
    {
        MessageBox.Show("Ошибка чтения файла.\n" +
                        exc.ToString(), "NkEdit",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

// сохранить документ
private int SaveDocument()
{
    int result = 0;

    if (fn == string.Empty)
    {
        // отобразить диалог Сохранить
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            // отобразить имя файла в заголовке окна
            fn = saveFileDialog1.FileName;
            this.Text = fn;
        }
        else result = -1;
    }

    // сохранить файл
    if (fn != string.Empty)
```

```
{  
    try  
    {  
        // получим информацию о файле fn  
        System.IO.FileInfo fi = new System.IO.FileInfo(fn);  
  
        // поток для записи (перезаписываем файл)  
        System.IO.StreamWriter sw = fi.CreateText();  
  
        // записываем данные  
        sw.WriteLine(textBox1.Text);  
  
        // закрываем поток  
        sw.Close();  
        result = 0;  
    }  
    catch (Exception exc)  
    {  
        MessageBox.Show(exc.ToString(), "NkEdit",  
                        MessageBoxButtons.OK, MessageBoxIcon.Error);  
    }  
}  
}  
  
// выбор в меню Файл команды Создать  
private void FileCreateToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    if (docChanged)  
    {  
        DialogResult dr;  
        dr = MessageBox.Show("Сохранить изменения?", "NkEdit",  
                            MessageBoxButtons.YesNoCancel,  
                            MessageBoxIcon.Warning);  
        switch (dr)  
        {  
            case DialogResult.Yes:  
                if (SaveDocument() == 0)  
                {  
                    textBox1.Clear();  
                    docChanged = false;  
                }  
                break;  
            case DialogResult.No:  
                textBox1.Clear();  
                docChanged = false;  
                break;  
        }  
    }  
}
```

```
        case DialogResult.Cancel:
            //
            break;
    };
}

// выбор в меню Файл команды Открыть
private void FileOpenToolStripMenuItem_Click(object sender, EventArgs e)
{
    openFileDialog1.FileName = string.Empty;

    // отобразить диалог Открыть
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        fn = openFileDialog1.FileName;

        // отобразить имя файла в заголовке окна
        this.Text = fn;

        try
        {
            // считываем данные из файла
            System.IO.StreamReader sr = new System.IO.StreamReader(fn);

            textBox1.Text = sr.ReadToEnd();
            textBox1.SelectionStart = textBox1.TextLength;

            sr.Close();
        }
        catch (Exception exc)
        {
            MessageBox.Show("Ошибка чтения файла.\n" +
                            exc.ToString(), "MEdit",
                            MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}

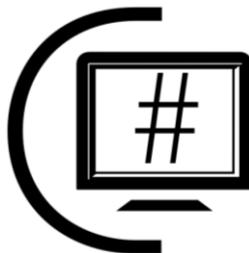
// выбор в меню Файл команды Сохранить
private void FileSaveToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveDocument();
}

// выбор в меню Файл команды Выход
private void FileExitToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{  
    this.Close();  
}  
  
// выбор в меню Параметры команды Панель инструментов  
private void ParamToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    // отобразить/скрыть панель инструментов  
    toolStrip1.Visible = ! toolStrip1.Visible;  
    ParamToolStripMenuItem.Checked = ! ParamToolStripMenuItem.Checked;  
}  
  
// выбор в меню Параметры команды Шрифт  
private void ParamFontToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    fontDialog1.Font = textBox1.Font;  
    if (fontDialog1.ShowDialog() == DialogResult.OK)  
    {  
        textBox1.Font = fontDialog1.Font;  
    }  
}  
  
private void textBox1_TextChanged(object sender, EventArgs e)  
{  
    docChanged = true;  
}  
  
// пользователь сделал щелчок на кнопке "Закрыть окно"  
private void Form1_FormClosing(object sender, FormClosingEventArgs e)  
{  
    if (docChanged)  
    {  
        DialogResult dr;  
        dr = MessageBox.Show("Сохранить изменения?", "NkEdit",  
                             MessageBoxButtons.YesNoCancel,  
                             MessageBoxIcon.Warning);  
        switch (dr)  
        {  
            case DialogResult.Yes :  
                if (SaveDocument() != 0)  
                    // пользователь отменил операцию сохранения файла  
                    e.Cancel = true; // отменить закрытие окна программы  
                break;  
            case DialogResult.No :  
                break;  
        }  
    }  
}
```

```
case DialogResult.Cancel:  
    // отменить закрытие окна программы  
    e.Cancel = true;  
    break;  
};  
}  
}  
  
// выбор в меню Справка команды О программе  
private void оПрограммаToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    Form2 about = new Form2();  
    about.ShowDialog();  
}  
  
// выбор в меню Файл команды Печать  
private void печатьToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    printDialog1.ShowDialog();  
}  
}
```

Visual

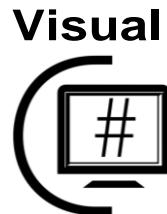


ЧАСТЬ II

Практикум программирования

В этой части рассматриваются задачи программирования графики, баз данных, разработки консольных приложений, создания справочной системы. Здесь же уделено внимание вопросам отладки и развертыванию приложений.

Глава 4.	Графика
Глава 5.	Базы данных
Глава 6.	Консольное приложение
Глава 7.	LINQ
Глава 8.	Отладка программы
Глава 9.	Справочная информация
Глава 10.	Публикация приложения
Глава 11.	Примеры программ
Глава 12.	Краткий справочник



ГЛАВА 4

Графика

В этой главе рассказывается, что надо сделать, чтобы на поверхности формы появилась иллюстрация, фотография или картинка, сформированная из графических примитивов (линий, прямоугольников, окружностей). Также рассматриваются принципы создания анимации.

Отображение графики обеспечивает компонент `PictureBox` (рис. 4.1). Подобно тому, как художник рисует на поверхности холста, программа "рисует" на *графической поверхности* компонента `PictureBox`.

Графическая поверхность компонента `PictureBox` представляет собой объект `Graphics`, методы которого и обеспечивают вывод графики. Таким образом, для того чтобы на поверхности компонента `PictureBox` появилась линия, прямоугольник, окружность или загруженная из файла иллюстрация, необходимо вызвать соответствующий метод объекта `Graphics`.

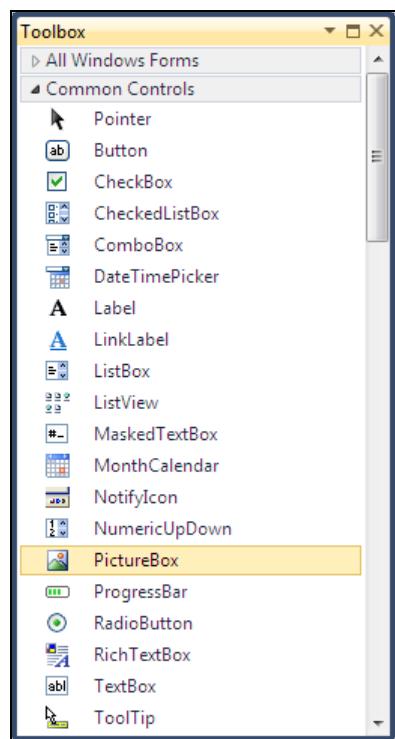


Рис. 4.1. Компонент `PictureBox`

Доступ к графической поверхности объекта (свойству `Graphics`) есть только у функции обработки события `Paint`. Поэтому сформировать (отобразить) графику может только она. Здесь следует вспомнить, что событие `Paint` возникает в начале работы программы, когда окно, а следовательно, и все компоненты, появляются на

экране. Событие Paint также возникает во время работы программы всякий раз, когда окно или его часть вновь появляется на экране, например, после того, как пользователь сдвинет другое окно, частично или полностью перекрывающее окно программы, или развернет свернутое окно.

Создается процедура обработки события Paint обычным образом: сначала надо выбрать компонент PictureBox, затем в окне **Properties** открыть вкладку **Events** и в поле события Paint ввести имя функции обработки события. Вместо ввода имени функции обработки события можно в поле имени функции сделать двойной щелчок левой кнопкой мыши. В этом случае имя функции обработки события сформирует среда разработки.

В качестве примера в листинге 4.1 приведена функция обработки события Paint компонента PictureBox, которая в поле компонента рисует итальянский флаг (рис. 4.2). Обратите внимание: доступ к объекту Graphics обеспечивает параметр e функции обработки события.

Листинг 4.1. Пример обработки события Paint

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    int x, y; // левый верхний угол полосы
    int w, h; // ширина и высота полосы

    int x0, y0; // левый верхний угол флага
    x0 = 10;
    y0 = 10;
    w = 28;
    h = 60;

    x = x0;
    y = y0;

    // зеленая полоса
    e.Graphics.FillRectangle(System.Drawing.Brushes.Green, x, y, w, h);
    // белая полоса
    x += w;
    e.Graphics.FillRectangle(System.Drawing.Brushes.White, x, y, w, h);
    // красная полоса
    x += w;
    e.Graphics.FillRectangle(System.Drawing.Brushes.Red, x, y, w, h);

    // контур
    e.Graphics.DrawRectangle(System.Drawing.Pens.Black, x0, y0, w*3, h);

    // подпись.
    // Используется шрифт, заданный свойством Font формы.
    e.Graphics.DrawString("Италия", this.Font,
        System.Drawing.Brushes.Black, x0, y0 + h + 10);
}
```

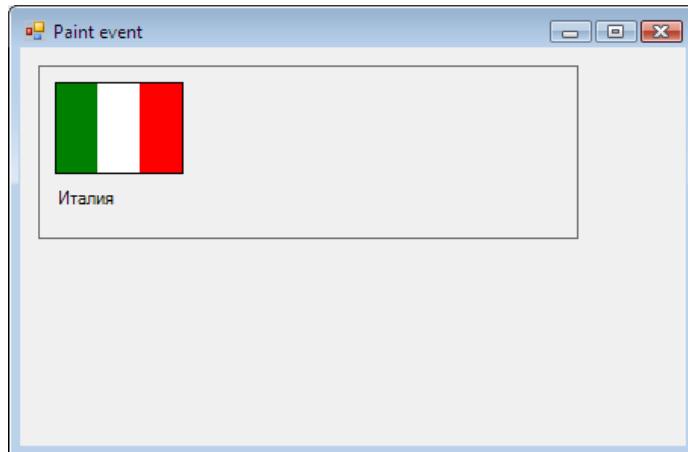


Рис. 4.2. Флаг нарисовала функция обработки события Paint

Графическая поверхность

Графическая поверхность состоит из отдельных точек — пикселов. Положение точки на графической поверхности характеризуется горизонтальной (x) и вертикальной (y) координатами (рис. 4.3). Координаты точек отсчитываются от левого верхнего угла и возрастают слева направо (координата x) и сверху вниз (координата y). Левая верхняя точка графической поверхности имеет координаты $(0, 0)$. Размер графической поверхности формы соответствует размеру *клиентской* области (т. е. без учета высоты области заголовка и ширины границ) формы (свойство

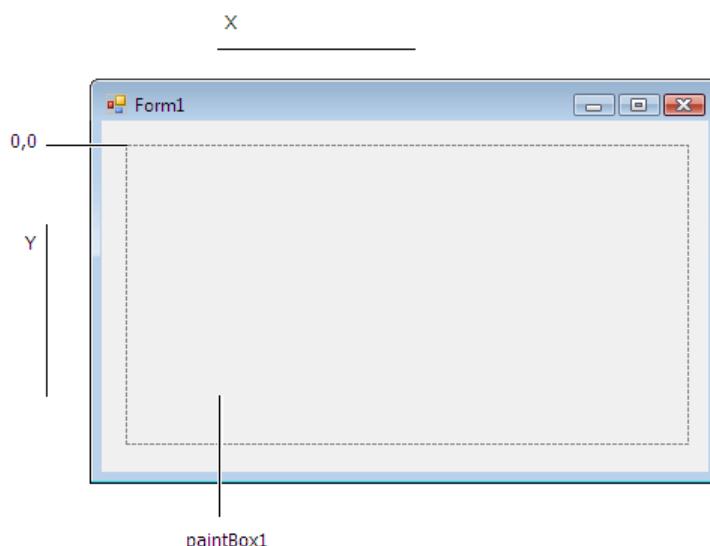


Рис. 4.3. Координаты точек графической поверхности формы

`ClientSize`), а размер графической поверхности компонента `PictureBox` — размеру компонента.

Карандаши и кисти

Методы рисования графических примитивов (например, `DrawLine` — линия, `DrawRectangle` — прямоугольник, `FillRectangle` — область) используют *карандаши* и *кисти*. Карандаш (объект `Pen`) определяет вид линии, кисть (объект `Brush`) — вид закраски области. Например, метод

```
DrawLine(System.Drawing.Pens.Black, 10, 20, 100, 20);
```

рисует из точки (10, 20) горизонтальную линию длиной 100 пикселов, используя черный (`Black`) карандаш из стандартного набора карандашей (`Pens`), а метод

```
FillRectangle(System.Drawing.Brushes.Green, 5, 10, 20, 20);
```

при помощи стандартной кисти зеленого (`Green`) цвета рисует зеленый квадрат, левый верхний угол которого находится в точке (5, 10).

Карандаши и кисти определены в пространстве имен `System.Drawing`.

Карандаш

Карандаш определяет вид линии — цвет, толщину и стиль. В распоряжении программиста есть два набора карандашей: стандартный и системный. Также программист может создать собственный карандаш.

Стандартный набор карандашей — это цветные карандаши (всего их 141), которые рисуют непрерывную линию толщиной в *один пиксель*. Некоторые карандаши из стандартного набора приведены в табл. 4.1.

Таблица 4.1. Некоторые карандаши из стандартного набора

Карандаш	Цвет
<code>Pens.Red</code>	Красный
<code>Pens.Orange</code>	Оранжевый
<code>Pens.Yellow</code>	Желтый
<code>Pens.Green</code>	Зеленый
<code>Pens.LightBlue</code>	Голубой
<code>Pens.Blue</code>	Синий
<code>Pens.Purple</code>	Пурпурный
<code>Pens.Black</code>	Черный
<code>Pens.LightGray</code>	Серый
<code>Pens.White</code>	Белый
<code>Pens.Transparent</code>	Прозрачный

Системный набор карандашей — это карандаши, цвет которых определяется текущей цветовой схемой операционной системы и совпадает с цветом какого-либо элемента интерфейса пользователя. Например, цвет карандаша SystemPens.ControlText совпадает с цветом, который в текущей цветовой схеме используется для отображения текста на элементах управления (командных кнопках и др.), а цвет карандаша SystemPens.WindowText — с цветом текста в окнах сообщений.

Карандаш из стандартного (`Pens`) и системного (`SystemPens`) наборов рисует непрерывную линию толщиной в *один пиксель*. Если надо нарисовать пунктирную линию или линию толщиной больше единицы, то следует использовать карандаш программиста.

Карандаш программиста — это объект типа `Pen`, свойства которого (табл. 4.2) определяют вид линии, рисуемой карандашом.

Таблица 4.2. Свойства объекта Pen

Свойство	Описание
<code>Color</code>	Цвет линии
<code>Width</code>	Толщина линии (задается в пикселях)
<code>DashStyle</code>	Стиль линии (<code>DashStyle.Solid</code> — сплошная; <code>DashStyle.Dash</code> — пунктирная, длинные штрихи; <code>DashStyle.Dot</code> — пунктирная, короткие штрихи; <code>DashStyle.DashDot</code> — пунктирная, чередование длинного и короткого штрихов; <code>DashStyle.DashDotDot</code> — пунктирная, чередование одного длинного и двух коротких штрихов; <code>DashStyle.Custom</code> — пунктирная линия, вид которой определяет свойство <code>DashPattern</code>)
<code>DashPattern</code>	Длина штрихов и промежутков пунктирной линии <code>DashStyle.Custom</code>

Для того чтобы использовать карандаш программиста, его надо создать. Создает карандаш конструктор объекта `Pen`. Конструктор *перегружаемый*, т. е. для объекта `Pen` определено несколько конструкторов, которые различаются количеством параметров. Например, конструктор `Pen(Цвет)` создает карандаш указанного цвета толщиной в один пиксель, а `Pen(Цвет, Толщина)` — карандаш указанного цвета и толщины. В качестве параметра `Цвет` можно использовать константу типа `Color` (табл. 4.3). Другие константы типа `Color` (а их более 100) можно найти в справочной системе.

Таблица 4.3. Константы типа Color

Константа	Цвет
<code>Color.Red</code>	Красный
<code>Color.Orange</code>	Оранжевый
<code>Color.Yellow</code>	Желтый
<code>Color.Green</code>	Зеленый
<code>Color.LightBlue</code>	Голубой

Таблица 4.3 (окончание)

Константа	Цвет
Color.Blue	Синий
Color.Purple	Пурпурный
Color.Black	Черный
Color.LightGray	Серый
Color.White	Белый
Color.Transparent	Прозрачный

Цвет, ширину линии и стиль карандаша, созданного программистом, можно изменить. Чтобы это сделать, надо изменить значение соответствующего свойства.

В листинге 4.2 приведена процедура, которая демонстрирует создание и использование карандаша программиста.

Листинг 4.2. Создание карандаша

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    System.Drawing.Pen aPen; // карандаш

    // создать красный "толстый" карандаш
    aPen = new System.Drawing.Pen(Color.Red, 2);

    e.Graphics.DrawRectangle(aPen, 10, 10, 100, 100);

    // теперь карандаш зеленый и его толщина 4 пикселя
    aPen.Width = 4;
    aPen.Color = Color.Green;

    // рисуем зеленым карандашом
    e.Graphics.DrawRectangle(aPen, 20, 20, 100, 100);

    // теперь линия пунктирная
    aPen.Width = 1;
    aPen.Color = Color.Purple;
    aPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;

    // рисуем пунктиром
    e.Graphics.DrawRectangle(aPen, 30, 30, 100, 100);
}
```

Программист может создать карандаш, который рисует пунктирную линию, отличную от стандартной. Чтобы это сделать, надо в свойство `DashPattern` поместить

ссылку на массив описания отрезка линии, а свойству `DashStyle` присвоить значение `DashStyle.Custom`.

Массив описания отрезка линии — это состоящий из двух элементов одномерный массив (типа `float`), который содержит информацию об отрезке пунктирной линии (цепочке "штрих — пропуск"). Первый элемент массива задает длину штриха, второй — пропуска. Приведенная в листинге 4.3 функция демонстрирует процесс создания карандаша, который рисует пунктирную линию, определенную программистом.

Листинг 4.3. Линия, определенная программистом

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    System.Drawing.Pen aPen; // карандаш

    // создать красный карандаш
    aPen = new System.Drawing.Pen(Color.Red, 1);

    // стиль линии, определенный программистом
    float[] myPattern;
    myPattern = new float[2];

    myPattern[0] = 15; // штрих
    myPattern[1] = 5; // пропуск

    aPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Custom;
    aPen.DashPattern = myPattern;

    e.Graphics.DrawRectangle(aPen, 40, 40, 100, 100); // прямоугольник
}
```

Кисть

Кисти используются для закраски внутренних областей геометрических фигур. Например, инструкция

```
e.Graphics.FillRectangle(Brushes.DeepSkyBlue, x, y, w, h);
```

рисует синий (темно-небесный) закрашенный прямоугольник.

В приведенном примере `Brushes.DeepSkyBlue` — это стандартная кисть темно-небесного цвета. Параметры `x`, `y` определяют положение прямоугольника; `w`, `h` — его размер.

В распоряжении программиста есть три типа кистей: стандартные, штриховые и текстурные.

Стандартная кисть закрашивает область одним цветом (сплошная закраска). Всего есть 140 кистей, некоторые из них приведены в табл. 4.4. Полный список кистей можно найти в справочной системе.

Таблица 4.4. Некоторые кисти из стандартного набора

Кисть	Цвет
Brushes.Red	Красный
Brushes.Orange	Оранжевый
Brushes.Yellow	Желтый
Brushes.Green	Зеленый
Brushes.LightBlue	Голубой
Brushes.Blue	Синий
Brushes.Purple	Пурпурный
Brushes.Black	Черный
Brushes.LightGray	Серый
Brushes.White	Белый
Brushes.Transparent	Прозрачный

Штриховая кисть (`HatchBrush`) закрашивает область путем штриховки. Область может быть заштрихована горизонтальными, вертикальными или наклонными линиями разного стиля и толщины. В табл. 4.5 перечислены некоторые из возможных стилей штриховки. Полный список стилей штриховки можно найти в справочной системе.

Таблица 4.5. Некоторые стили штриховки областей

Стиль	Штриховка
<code>HatchStyle.LightHorizontal</code>	Редкая горизонтальная
<code>HatchStyle.Horizontal</code>	Средняя горизонтальная
<code>HatchStyle.NarrowHorizontal</code>	Частая горизонтальная
<code>HatchStyle.LightVertical</code>	Редкая вертикальная
<code>HatchStyle.Vertical</code>	Средняя вертикальная
<code>HatchStyle.NarrowVertical</code>	Частая вертикальная
<code>HatchStyle.LargeGrid</code>	Крупная сетка из горизонтальных и вертикальных линий
<code>HatchStyle.SmallGrid</code>	Мелкая сетка из горизонтальных и вертикальных линий
<code>HatchStyle.DottedGrid</code>	Сетка из горизонтальных и вертикальных линий, составленных из точек
<code>HatchStyle.ForwardDiagonal</code>	Диагональная штриховка "вперед"
<code>HatchStyle.BackwardDiagonal</code>	Диагональная штриховка "назад"
<code>HatchStyle.Percent05 — HatchStyle.Percent90</code>	Точки (степень заполнения 5%, 10%, ..., 90%)

Таблица 4.5 (окончание)

Стиль	Штриховка
HatchStyle.HorizontalBrick	"Кирпичная стена"
HatchStyle.LargeCheckerBoard	"Шахматная доска"
HatchStyle.SolidDiamond	"Бриллиант" ("Шахматная доска", повернутая на 45°)
HatchStyle.Sphere	"Пузырьки"
HatchStyle.ZigZag	"Зигзаг"

В листинге 4.4 приведена функция, демонстрирующая процесс создания и использования штриховой кисти. При создании кисти конструктору передаются: константа HatchStyle, которая задает вид штриховки, и две константы типа Color, первая из которых определяет цвет штрихов, вторая — цвет фона. Следует обратить внимание, класс HatchBrush определен в пространстве имен System.Drawing.Drawing2D, поэтому в программу, использующую штриховые кисти, надо добавить ссылку (директиву using) на это пространство имен.

Листинг 4.4. Пример создания и использования штриховой кисти

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    // штриховка (HatchBrush-кисть)
    System.Drawing.Drawing2D.HatchBrush myBrush;

    myBrush = new System.Drawing.Drawing2D.HatchBrush(
        HatchStyle.DottedGrid, // стиль штриховки
        Color.Black,          // цвет штрихов
        Color.SkyBlue);       // цвет фона

    e.Graphics.FillRectangle(myBrush, 10, 10, 190, 160);
}
```

Текстурная кисть (TextureBrush) представляет собой рисунок, который обычно загружается во время работы программы из файла (bmp, jpg или gif) или из ресурса. Закраска области текстурной кистью выполняется путем дублирования рисунка внутри области.

Листинг 4.5 демонстрирует процесс создания и использования текстурной кисти. Кисть создает конструктор, которому в качестве параметра передается текстура — имя графического файла.

Листинг 4.5. Пример создания и использования текстурной кисти

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    float x,y,w,h;
```

```
x = 20;
y = 20;
w = 90;
h = 60;

System.Drawing.TextureBrush myBrush; // текстурная кисть

try
{
    // загрузить текстуру из файла
    myBrush = new System.Drawing.TextureBrush
        (Image.FromFile(Application.StartupPath+"\brush_1.bmp"));

    // рисуем текстурной кистью
    e.Graphics.FillRectangle(myBrush, x, y, w, h);
}
catch (System.IO.FileNotFoundException ex)
{
    e.Graphics.DrawRectangle(Pens.Black, x, y, w, h);
    e.Graphics.DrawString("Source image",
        this.Font, Brushes.Black, x+5, y+5);
    e.Graphics.DrawString(" not found",
        this.Font, Brushes.Black, x+5, y+20);
}
```

Возможна градиентная, в простейшем случае двухцветная, закраска области, при которой цвет закраски области меняется от одного к другому вдоль линии градиента.

Чтобы закрасить область градиентом, надо создать *градиентную* кисть. При создании кисти конструктору передаются линия градиента и две константы, определяющие цвет градиента. Например:

```
myBrush = new System.Drawing.Drawing2D.LinearGradientBrush
    (Point(10,10), Point(110,10), Color.Red, Color.White);
```

После того как градиентная кисть будет создана, ее можно использовать для закраски области:

```
e.Graphics.FillRectangle(myBrush, 10, 10, 100, 40);
```

Следует обратить внимание, для равномерной закраски области координаты точек линии градиента должны находиться на границе закрашиваемой области или быть вне ее.

Иногда возникает необходимость, используя один и тот же градиент, закрасить несколько областей разного размера. Если попытаться закрасить область большего размера градиентной кистью меньшего размера, то область большего размера будет выглядеть так, как показано на рис. 4.4.

Чтобы область была закрашена равномерно, для нее надо создать свою (новую) градиентную кисть или трансформировать существующую. Приведенная в листин-

где 4.6 функция показывает, как это сделать. Она же показывает, как можно изменить цвет градиентной кисти.



Размер градиентной кисти соответствует размеру области



Размер градиентной кисти не соответствует размеру области

Рис. 4.4. Закраска области градиентной кистью

Листинг 4.6. Пример создания и изменения градиентной кисти

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    // положение и размер области, которую надо закрасить
    // градиентно (градиентной кистью)
    Rectangle myRect = new Rectangle(20, 10, 200, 40);

    // создать градиентную кисть
    System.Drawing.Drawing2D.LinearGradientBrush myBrush =
        new System.Drawing.Drawing2D.LinearGradientBrush
            (myRect, Color.Red, Color.White, 0.0f, true);

    // закрасить область градиентно
    e.Graphics.FillRectangle(myBrush, myRect);

    // изменить цвет градиента
    Color[] nc; // nc - NewColor

    nc = new Color[2];
    nc[0] = Color.Green;
    nc[1] = Color.Yellow;

    myBrush.LinearColors = nc;

    // Так как будем закрашивать область другого размера,
    // то надо изменить размер градиентной кисти
    Point[] transformArray;

    transformArray = new Point[3];

    transformArray[0] = new Point(20, 60);
    transformArray[1] = new Point(420, 60);
    transformArray[2] = new Point(20, 100);
```

```

Matrix myMatrix = new Matrix(myRect, transformArray);

myBrush.MultiplyTransform(myMatrix, MatrixOrder.Prepend);

// рисуем новой кистью
e.Graphics.FillRectangle(myBrush, 20, 60, 400, 40);

// восстановить размер кисти
myBrush.ResetTransform();

// рисуем область такого же размера, как и первую
e.Graphics.FillRectangle(myBrush, 20, 120, 200, 40);
}

```

Графические примитивы

Любая картинка, чертеж, схема представляет собой совокупность графических **примитивов**: точек, линий, окружностей, дуг, текста и др.

Вычерчивание графических примитивов на графической поверхности (`Graphics`) выполняют соответствующие методы (табл. 4.6).

Таблица 4.6. Некоторые методы вычерчивания графических примитивов

Метод	Действие
<code>DrawLine(Pen, x1, y1, x2, y2)</code> <code>DrawLine(Pen, p1, p2)</code>	Рисует линию. Параметр <code>Pen</code> определяет цвет, толщину и стиль линии; параметры <code>x1, y1, x2, y2</code> или <code>p1</code> и <code>p2</code> — координаты точек начала и конца линии
<code>DrawRectangle(Pen, x, y, w, h)</code>	Рисует контур прямоугольника. Параметр <code>Pen</code> определяет цвет, толщину и стиль границы прямоугольника: параметры <code>x, y</code> — координаты левого верхнего угла; параметры <code>w</code> и <code>h</code> задают размер прямоугольника
<code>FillRectangle(Brush, x, y, w, h)</code>	Рисует закрашенный прямоугольник. Параметр <code>Brush</code> определяет цвет и стиль закраски прямоугольника; параметры <code>x, y</code> — координаты левого верхнего угла; параметры <code>w</code> и <code>h</code> задают размер прямоугольника
<code>DrawEllipse(Pen, x, y, w, h)</code>	Рисует эллипс (контур). Параметр <code>Pen</code> определяет цвет, толщину и стиль линии эллипса; параметры <code>x, y, w, h</code> — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
<code>FillEllipse(Brush, x, y, w, h)</code>	Рисует закрашенный эллипс. Параметр <code>Brush</code> определяет цвет и стиль закраски внутренней области эллипса; параметры <code>x, y, w, h</code> — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс

Таблица 4.6 (окончание)

Метод	Действие
DrawPolygon(Pen, P)	Рисует контур многоугольника. Параметр Pen определяет цвет, толщину и стиль линии границы многоугольника; параметр P (массив типа Point) — координаты углов многоугольника
FillPolygon(Brush, P)	Рисует закрашенный многоугольник. Параметр Brush определяет цвет и стиль закраски внутренней области многоугольника; параметр P (массив типа Point) — координаты углов многоугольника
DrawString(str, Font, Brush, x, y)	Выводит на графическую поверхность строку текста. Параметр Font определяет шрифт; Brush — цвет символов; x и y — точку, от которой будет выведен текст
DrawImage(Image, x, y)	Выводит на графическую поверхность иллюстрацию. Параметр Image определяет иллюстрацию; x и y — координату левого верхнего угла области вывода иллюстрации

Один и тот же элемент можно нарисовать при помощи *разных*, но имеющих *одинаковые* имена методов (вспомните: возможность объявления функций, имеющих одинаковые имена, но разные параметры, называется *перегрузкой*).

Например, прямоугольник можно нарисовать методом `DrawRectangle`, которому в качестве параметров передаются координаты левого верхнего угла и размеры прямоугольника:

```
e.Graphics.DrawRectangle(Pens.Black, x, x, w, h)
```

Эту же задачу может решить метод `DrawRectangle`, которому в качестве параметра передается структура типа `Rectangle`, поля которой определяют прямоугольник (положение и размер):

```
Rectangle aRect = new Rectangle(20,100,50,50);
e.Graphics.DrawRectangle(Pens.Blue, aRect);
```

Существование нескольких методов, выполняющих одну и ту же задачу, позволяет программисту выбрать метод, наиболее подходящий для решения конкретной задачи.

В качестве параметров методов вычерчивания графических примитивов часто используется структура `Point`. Ее поля `x` и `y` определяют положение (координаты) точки графической поверхности. Например:

```
Point p1 = new Point(10,10);
Point p2 = new Point(100,10);
// рисуем линию из p1 в p2
e.Graphics.DrawLine(Pens.Green, p1, p2);
```

Линии

Метод `DrawLine` рисует прямую линию. В инструкции вызова метода следует указать карандаш, которым надо нарисовать линию, и координаты точек начала и конца линии:

```
DrawLine(aPen, x1, y1, x2, y2)
```

или

```
DrawLine(aPen, p1, p2)
```

Параметр `aPen` (типа `Brush`) задает карандаш, которым рисуется линия, `x1` и `y1` или `p1` — точку начала линии, а `x2` и `y2` или `p2` — точку конца линии. Параметры `x1`, `y1`, `x2` и `y2` должны быть одного типа (`Integer` или `Single`). Тип параметров `p1` и `p2` — `Point`.

Например, инструкция

```
e.Graphics.DrawLine(Pens.Green, 10,10,300,10);
```

рисует зеленую линию толщиной в один пиксель из точки (10, 10) в точку (300, 10).

Эту же линию можно нарисовать и так:

```
Point p1 = new Point(10,10);
Point p2 = new Point(300,10);
e.Graphics.DrawLine(Pens.Green, p1, p2);
```

Программа "График" (ее окно приведено на рис. 4.5, а текст — в листинге 4.7) показывает использование метода `DrawLine`. Данные, отображаемые в окне программы, загружаются из текстового файла.

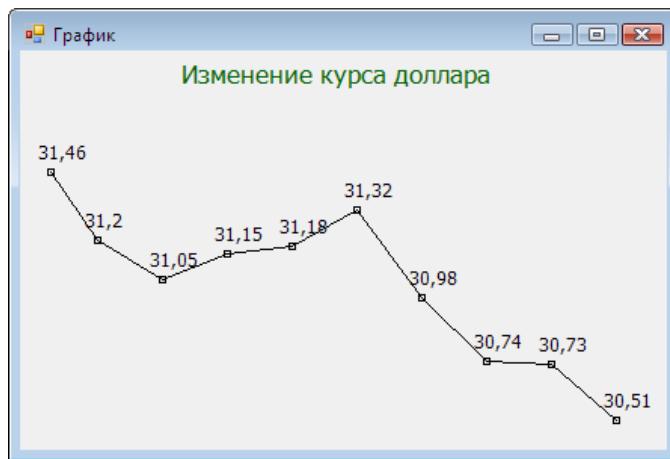


Рис. 4.5. Программа "График"

Загружает данные из файла конструктор формы (файл данных должен находиться в папке приложения, при запуске программы в режиме отладки из среды разработки — в папке `Debug`). Он же выполняет обработку массива данных — ищет минимальный и максимальный элементы ряда. Конструктор также, в случае успешной

загрузки данных, задает функцию обработки события `Paint` формы. Непосредственно вывод графика на поверхность формы выполняет функция `drawDiagram`. Ее вызывает функция обработки события `Paint` формы, а также косвенно (путем вызова метода `Refresh`) — функция обработки события `Resize` формы.

Листинг 4.7. График

```
// данные
private double[] d;

// строит график
private void drawDiagram(object sender, PaintEventArgs e)
{
    // графическая поверхность
    Graphics g = e.Graphics;

    // шрифт подписей данных
    Font dFont = new Font("Tahoma", 9);

    // шрифт заголовка
    Font hFont = new Font("Tahoma", 14, FontStyle.Regular);
    string header = "Курс доллара";

    // ширина области отображения текста
    int w = (int)g.MeasureString(header, hFont).Width;

    int x = (this.ClientSize.Width - w) / 2;

    g.DrawString(header, hFont, System.Drawing.Brushes.ForestGreen, x, 5);

    /*
        Область построения графика:
        - отступ сверху - 100;
        - отступ снизу - 20;
        - отступ слева - 20;
        - отступ справа - 20.

        ClientSize - размер внутренней области окна

        График строим в отклонениях от минимального значения ряда данных
        так, чтобы он занимал всю область построения.
    */

    // расстояние между точками графика (шаг по X)
    int sw = (int)((this.ClientSize.Width - 40) / (d.Length - 1));

    double max = d[0]; // максимальный элемент массива
    double min = d[0]; // минимальный элемент массива
```

```
for (int i = 1; i < d.Length; i++)
{
    if (d[i] > max) max = d[i];
    if (d[i] < min) min = d[i];
}

// рисуем график
int x1, y1, x2, y2;

// первая точка
x1 = 20;
y1 = this.ClientSize.Height - 20 -
    (int)((this.ClientSize.Height - 100)*(d[0] - min) / (max - min));

// маркер первой точки
g.DrawRectangle(System.Drawing.Pens.Black, x1 - 2, y1 - 2, 4, 4);

// подпись численного значения первой точки
g.DrawString(Convert.ToString(d[0]),
    dFont, System.Drawing.Brushes.Black, x1 - 10, y1 - 20);

// остальные точки
for (int i = 1; i < d.Length; i++)
{
    x2 = 8 + i * sw;
    y2 = this.ClientSize.Height - 20 -
        (int)((this.ClientSize.Height - 100)*(d[i] - min) / (max - min));

    // маркер точки
    g.DrawRectangle(System.Drawing.Pens.Black, x2 - 2, y2 - 2, 4, 4);

    // соединим текущую точку с предыдущей
    g.DrawLine(System.Drawing.Pens.Black, x1, y1, x2, y2);

    // подпись численного значения
    g.DrawString(Convert.ToString(d[i]),
        dFont, System.Drawing.Brushes.Black, x2 - 10, y2 - 20);

    x1 = x2;
    y1 = y2;
}

public Form1()
{
    InitializeComponent();
}
```

```
// чтение данных из файла в массив
System.IO.StreamReader sr; // поток для чтения
try
{
    // Создаем поток для чтения.
    // Application.StartupPath возвращает путь к каталогу,
    // из которого была запущена программа
    sr = new System.IO.StreamReader(
        Application.StartupPath + "\\usd.dat");

    // создать массив
    d = new double[10];

    // читаем данные из файла в массив
    int i = 0;
    string t = sr.ReadLine();
    while ((t != null) && (i < d.Length))
    {
        // записываем считанное число в массив
        d[i++] = Convert.ToDouble(t);
        t = sr.ReadLine();
    }

    // закрываем поток
    sr.Close();

    // задаем функцию обработки события Paint
    this.Paint += new PaintEventHandler(drawDiagram);
}

// обработка исключений:

// - файл данных не найден
catch (System.IO.FileNotFoundException ex)
{
    MessageBox.Show(ex.Message + "\n" +
        "(" + ex.GetType().ToString() + ")",
        "График", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

// - другие исключения
catch (Exception ex)
{
    MessageBox.Show(ex.ToString(), "",
        MessageBoxButtons.OK, MessageBoxIcon.Stop);
}
```

```
private void Form1_SizeChanged(object sender, EventArgs e)
{
    this.Refresh();
}
```

Ломаная линия

Метод `DrawLines` рисует ломаную линию. В качестве параметров методу передается карандаш (`Pen`) и массив типа `Point`, элементы которого содержат координаты узловых точек линии. Метод рисует ломаную линию, последовательно соединяя точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д. Например, следующий фрагмент кода рисует ломаную линию, состоящую из четырех звеньев.

```
Point[] p; // массив точек

p = new Point[5];

// задать координаты точек кривой
p[0].X = 10; p[0].Y = 50;
p[1].X = 20; p[1].Y = 20;
p[2].X = 30; p[2].Y = 50;
p[3].X = 40; p[3].Y = 20;
p[4].X = 50; p[4].Y = 50;

// рисовать ломаную линию
e.Graphics.DrawLines(Pens.Green, p);
```

Метод `DrawLines` можно использовать для вычерчивания замкнутых контуров. Для этого первый и последний элементы массива должны содержать координаты одной и той же точки.

Прямоугольник

Метод `DrawRectangle` чертит прямоугольник (рис. 4.6). В качестве параметров метода надо указать карандаш, координаты левого верхнего угла и размер прямоугольника:

```
DrawRectangle(aPen, x, y, w, h);
```

Вместо четырех параметров, определяющих прямоугольник, можно указать структуру типа `Rectangle`:

```
DrawRectangle(aPen, aRect);
```

Поля `x` и `y` структуры `aRect` задают координаты левого верхнего угла прямоугольника, `aWidth` и `aHeight` — размер (ширину и высоту).

Вид линии границы прямоугольника (цвет, стиль и ширину) определяет параметр `aPen`, в качестве которого можно использовать один из стандартных карандашей или карандаш, созданный программистом.

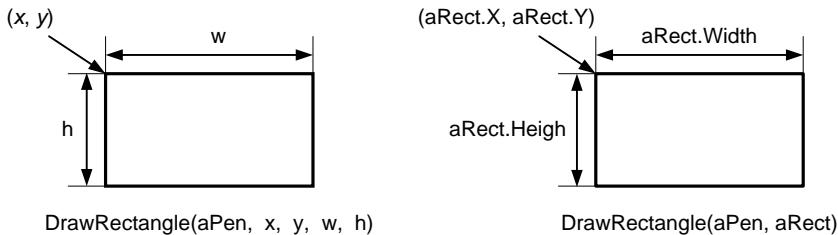


Рис. 4.6. Метод `DrawRectangle` рисует прямоугольник

Метод `FillRectangle` рисует закрашенный прямоугольник. В качестве параметров методу надо передать кисть, координаты левого верхнего угла и размер прямоугольника:

```
FillRectangle(aBrush, x, y, w, h);
```

Вместо x, y, w и h можно указать структуру типа `Rectangle`:

```
FillRectangle(aBrush, aRect);
```

Параметр `aBrush`, в качестве которого можно использовать стандартную или созданную программистом штриховую (`HatchBrush`), градиентную (`LinearGradientBrush`) или текстурную (`TextureBrush`) кисть, определяет цвет и стиль закраски области.

Далее приведен фрагмент кода (функции обработки события `Paint`), который демонстрирует использование методов `DrawRectangle` и `FillRectangle`.

```
Rectangle aRect; // положение и размер прямоугольника

// Зеленый прямоугольник размером 60x30,
// левый верхний угол которого в точке (10, 10)
aRect = new Rectangle(10, 10, 60, 30); // положение и размер
e.Graphics.FillRectangle(Brushes.ForestGreen, aRect);

// Желтый прямоугольник с черной границей размером 60x30,
// левый верхний угол которого в точке (100, 10)
aRect.X = 100;
e.Graphics.FillRectangle(Brushes.Gold, aRect); // прямоугольник
e.Graphics.DrawRectangle(Pens.Black, aRect); // граница
```

Точка

Казалось бы, чего проще — нарисовать на графической поверхности точку. Но у объекта `Graphics` нет метода, который позволяет это сделать. Метод `SetPixel` есть у объекта `Bitmap`. Поэтому если действительно необходимо сформировать картинку из точек, придется создать объект `Bitmap`, сформировать на его поверхности изображение, а затем это изображение при помощи метода `DrawImage` вывести на графическую поверхность. Но можно поступить проще — вместо точки вывести квадрат размером в один пиксель.

Например, инструкция

```
e.Graphics.FillRectangle(Brushes.Red, x, y, 1, 1);
```

рисует на графической поверхности красную точку.

Многоугольник

Метод `DrawPolygon` чертит многоугольник (контур). Инструкция вызова метода в общем виде выглядит так:

```
DrawPolygon(aPen, p)
```

Параметр `p` — массив типа `Point`, определяет координаты вершин многоугольника. Метод `DrawPolygon` чертит многоугольник, соединяя прямыми линиями точки, координаты которых находятся в массиве: первую со второй, вторую с третьей и т. д. Последняя точка соединяется с первой. Вид границы многоугольника определяет параметр `aPen`, в качестве которого можно использовать стандартный или созданный программистом карандаш.

Закрашенный многоугольник рисует метод `FillPolygon`. Инструкция вызова метода в общем виде выглядит так:

```
FillPolygon(aBrush, p)
```

Параметр `aBrush`, в качестве которого можно использовать стандартную или созданную программистом штриховую (`HatchBrush`), градиентную (`LinearGradientBrush`) или текстурную (`TextureBrush`) кисть, определяет цвет и стиль закраски внутренней области многоугольника.

Далее приведен фрагмент кода, который демонстрирует использование методов `DrawPolygon` и `FillPolygon` — рисует корону.

```
Point[] p;
```

```
p = new Point[5];
```

```
p[0].X = 10; p[0].Y = 30;
p[1].X = 10; p[1].Y = 10;
p[2].X = 30; p[2].Y = 20;
p[3].X = 50; p[3].Y = 10;
p[4].X = 50; p[4].Y = 30;
```

```
e.Graphics.FillPolygon(Brushes.Gold, p);
e.Graphics.DrawPolygon(Pens.Black, p);
```

Эллипс и окружность

Метод `DrawEllipse` чертит эллипс внутри прямоугольной области (рис. 4.7). Если прямоугольник является квадратом, то метод рисует окружность.

Инструкция вызова метода `DrawEllipse` в общем виде выглядит так:

```
DrawEllipse(aPen, x, y, w, h);
```

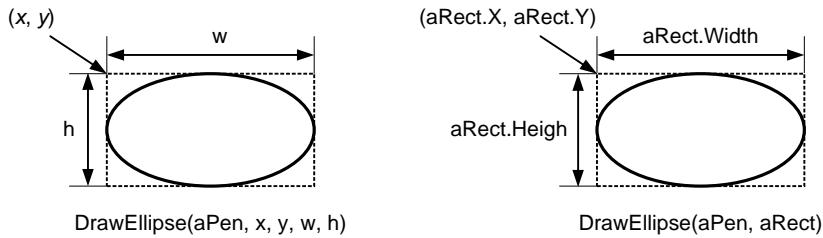


Рис. 4.7. Метод `DrawEllipse` рисует эллипс

Параметр `aPen`, в качестве которого можно использовать один из стандартных карандашей или карандаш, созданный программистом, определяет вид (цвет, толщину, стиль) границы эллипса. Параметры `x`, `y`, `w` и `h` задают координаты левого верхнего угла и размер прямоугольника, внутри которого метод рисует эллипс.

В инструкции вызова метода `DrawEllipse` вместо параметров `x`, `y`, `w` и `h` можно указать структуру типа `Rectangle`:

```
DrawEllipse(aPen, aRect);
```

Поля `x` и `y` структуры `aRect` задают координаты левого верхнего угла области, внутри которой метод рисует эллипс, а поля `Width` и `Height` — размер.

Метод `FillEllipse` рисует закрашенный эллипс. В инструкции вызова метода следует указать кисть (стандартную или созданную программистом), координаты и размер прямоугольника, внутри которого надо нарисовать эллипс:

```
FillEllipse(aBrush, x, y, w, h);
```

Кисть определяет цвет и способ закраски внутренней области эллипса.

Вместо параметров `x`, `y`, `w` и `h` можно указать структуру типа `Rectangle`:

```
FillEllipse(aBrush, aRect);
```

Дуга

Метод `DrawArc` рисует дугу — часть эллипса (рис. 4.8). Инструкция вызова метода в общем виде выглядит так:

```
DrawArc(aPen, x, y, w, h, startAngle, sweepAngle)
```

Параметры `x`, `y`, `w` и `h` определяют эллипс (окружность), частью которого является дуга. Параметр `startAngle` задает начальную точку дуги — пересечение эллипса и прямой, проведенной из центра эллипса и образующей угол `startAngle` с горизонтальной осью эллипса (угловая координата возрастает по часовой стрелке). Параметр `sweepAngle` задает длину дуги (в градусах). Если значение `sweepAngle` положительное, то дуга рисуется от начальной точки по часовой стрелке, если отрицательное — то против. Величины углов задаются в градусах.

В инструкции вызова метода `DrawArc` вместо параметров `x`, `y`, `w` и `h` можно указать структуру типа `Rectangle`:

```
DrawArc(aPen, aRect, startAngle, sweepAngle)
```

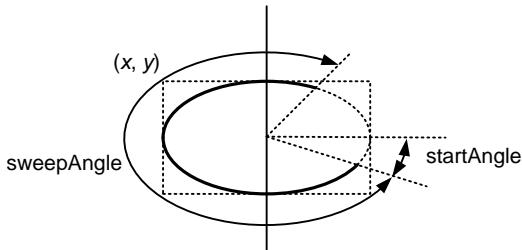


Рис. 4.8. Значения параметров метода `DrawArc` определяют дугу как часть эллипса

Сектор

Метод `DrawPie` рисует границу сектора (рис. 4.9). Инструкция вызова метода выглядит так:

```
DrawPie(aPen, x, y, w, h, startAngle, sweepAngle);
```

Параметры `x`, `y`, `w` и `h` определяют эллипс, частью которого является сектор. Параметр `startAngle` задает начальную точку дуги сектора — пересечение эллипса и прямой, проведенной из центра эллипса и образующей угол `startAngle` с горизонтальной осью эллипса (угловая координата возрастает по часовой стрелке). Параметр `sweepAngle` — длину дуги сектора (в градусах). Если значение `sweepAngle` положительное, то дуга сектора рисуется от начальной точки по часовой стрелке, если отрицательное — против. Величины углов задаются в градусах.

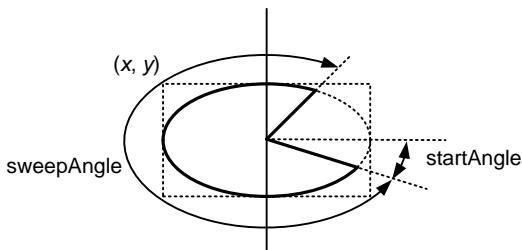


Рис. 4.9. Значения параметров метода `DrawPie` определяют сектор как часть эллипса

В инструкции вызова метода `DrawPie` вместо параметров `x`, `y`, `w` и `h` можно указать структуру типа `Rectangle`:

```
DrawPie(aPen, aRect, startAngle, sweepAngle)
```

Метод `FillPie` рисует сектор. Параметры у метода `FillPie`, за исключением первого, вместо которого надо указать кисть, такие же, как и у метода `DrawPie`.

Программа "Круговая диаграмма" демонстрирует использование методов `DrawPie` и `FillPie`. В ее окне в виде круговой диаграммы отображается результат опроса. Исходные данные для построения диаграммы (вопрос, варианты ответа, количество ответов на каждый из вопросов) загружаются из файла. Окно программы приведено на рис. 4.10, текст программы и пример файла данных — в листингах 4.8 и 4.9.

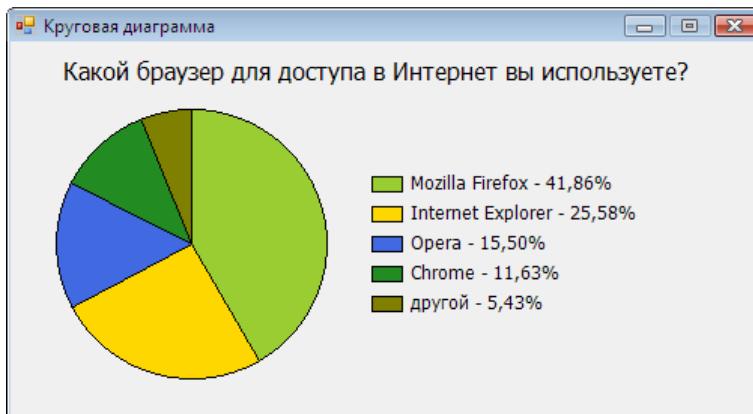


Рис. 4.10. Круговая диаграмма

Листинг 4.8. Круговая диаграмма

```
// заголовок диаграммы
string header;

// количество элементов данных
int N = 0;

double[] dat; // ряд данных
double[] p; // доля категории в общей сумме

// подписи данных
private string[] title;

public Form1()
{
    InitializeComponent();

    try
    {
        System.IO.StreamReader sr;

        sr = new System.IO.StreamReader(Application.StartupPath + "\\date.dat",
                                         System.Text.Encoding.GetEncoding(1251));

        // считываем заголовок диаграммы
        header = sr.ReadLine();

        // считываем данные о количестве записей и инициализируем массивы
        N = Convert.ToInt16(sr.ReadLine());
        dat = new double[N];
    }
}
```

```
p = new double[N];
title = new string[N];

// читаем данные
int i = 0;
string st;
st = sr.ReadLine();
while ((st != null) && (i < N))
{
    title[i] = st;
    st = sr.ReadLine();
    dat[i++] = Convert.ToDouble(st);
    st = sr.ReadLine();
}

// закрываем поток
sr.Close();

// задать процедуру обработки события Paint
this.Paint += new PaintEventHandler(Diagram);

double sum = 0;
int j = 0;

// вычислить сумму
for (j = 0; j < N; j++)
    sum += dat[j];

// вычислить долю каждой категории
for (j = 0; j < N; j++)
    p[j] = (double)(dat[j] / sum);
}

catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Диаграмма",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
}

// Процедура рисует круговую диаграмму;
// имеет такие же параметры, что и процедура обработки события Paint
private void Diagram(object sender, PaintEventArgs e)
{
    // графическая поверхность
    Graphics g = e.Graphics;
```

```
// шрифт заголовка
Font hFont = new Font("Tahoma", 12);

// выводим заголовок
int w = (int)g.MeasureString(header, hFont).Width;
int x = (this.ClientSize.Width - w) / 2;

g.DrawString(header, hFont, System.Drawing.Brushes.Black, x, 10);

// шрифт легенды
Font lFont = new Font("Tahoma", 9);

// диаметр диаграммы
int d = ClientSize.Height - 70;

int x0 = 30;
int y0 = (ClientSize.Height - d) / 2 + 10;

// координаты верхнего левого угла области легенды
int lx = 60 + d;
int ly = y0 + (d - N * 20 + 10) / 2;

// длина дуги сектора
int swe;

// кисть для заливки сектора диаграммы
Brush fBrush = Brushes.White;

// начальная точка дуги сектора
int sta = -90;

// рисуем диаграмму
for (int i = 0; i < N; i++)
{
    // длина дуги
    swe = (int)(360 * p[i]);

    // задать цвет сектора
    switch (i)
    {
        case 0:
            fBrush = Brushes.YellowGreen;
            break;
        case 1:
            fBrush = Brushes.Gold;
            break;
    }
}
```

```
case 2:  
    fBrush = Brushes.RoyalBlue;  
    break;  
case 3:  
    fBrush = Brushes.ForestGreen;  
    break;  
case 4:  
    fBrush = Brushes.Olive;  
    break;  
case 5:  
    fBrush = Brushes.SkyBlue;  
    break;  
case 6:  
    fBrush = Brushes.SteelBlue;  
    break;  
case 7:  
    fBrush = Brushes.Chocolate;  
    break;  
case 8:  
    fBrush = Brushes.LightGray;  
    break;  
case 9:  
    fBrush = Brushes.Gold; break;  
}  
  
// из-за округления возможна ситуация, при которой  
// будет промежуток между последним и первым секторами  
if (i == N - 1)  
{  
    // последний сектор  
    swe = 270 - sta;  
}  
  
// рисуем сектор  
g.FillPie(fBrush, x0, y0, d, d, sta, swe);  
  
// рисуем границу сектора  
g.DrawPie(System.Drawing.Pens.Black, x0, y0, d, d, sta, swe);  
  
// прямоугольник легенды  
g.FillRectangle(fBrush, lx, ly + i * 20, 20, 10);  
g.DrawRectangle(System.Drawing.Pens.Black, lx, ly + i * 20, 20, 10);  
  
// подпись рядом с прямоугольником  
g.DrawString(title[i] + " - " + p[i].ToString("P"),  
            lFont, System.Drawing.Brushes.Black,  
            lx + 24, ly + i * 20 - 3);
```

```
// начальная точка дуги для следующего сектора
sta = sta + swe;
}
}
```

Листинг 4.9. Пример файла данных для программы "Круговая диаграмма"

Какой браузер для доступа в Интернет вы используете?

5	Mozilla Firefox
54	Internet Explorer
33	Opera
20	Chrome
15	
другой	
7	

Текст

Вывод текста на графическую поверхность выполняет метод DrawString. В инструкции вызова метода указывается строка, шрифт, кисть и координаты точки, от которой надо вывести текст:

```
DrawString(st, aFont, aBrush, x, y);
```

Параметр st задает текст, параметр aFont — шрифт, который используется для отображения текста, а aBrush — цвет текста. Параметры x и y определяют координаты левого верхнего угла области отображения текста (рис. 4.11).

В программе, окно которой показано, текст выводит функция обработки события Paint формы.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    String st = "Microsoft Visual Studio 2010";
    e.Graphics.DrawString(st, this.Font, Brushes.Black, 20, 20);
}
```



Рис. 4.11. Координаты и размер области отображения текста

В приведенном примере для вывода текста используется шрифт формы, заданный свойством `Font`. Если текст надо вывести шрифтом, отличным от шрифта, заданного для формы, то этот шрифт следует создать — объявить и инициализировать объект типа `Font`.

Инструкция создания шрифта (вызыва конструктора) выглядит так:

```
System.Drawing.Font aFont =
    new System.Drawing.Font(FontFamily, Size, FontStyle);
```

Параметр `FontFamily` (строкового типа) задает шрифт, на основе которого создается новый (определяет семейство, к которому относится создаваемый шрифт). В качестве значения параметра `FontFamily` можно использовать название шрифта, зарегистрированного в системе (`Arial`, `Times New Roman`, `Tahoma`). Параметр `Size` задает размер (в пунктах) шрифта. Параметр `FontStyle` определяет стиль символов шрифта (`FontStyle.Bold` — полужирный; `FontStyle.Italic` — курсив; `FontStyle.UnderLine` — подчеркнутый). Параметр `FontStyle` можно не указывать. В этом случае будет создан шрифт обычного начертания (`FontStyle.Regular`).

Следует обратить внимание, что изменить характеристики созданного шрифта нельзя (свойства `FontFamily`, `Size` и `Style` объекта `Font` определены "только для чтения"). Поэтому если в программе предполагается использовать разные шрифты, их необходимо создать.

В листинге 4.10 приведена функция обработки события `Paint`, демонстрирующая создание и использование шрифтов для вывода текста на поверхность формы (рис. 4.12).



Рис. 4.12. Вывод текста на поверхность формы методом `DrawString`

Листинг 4.10. Создание и использование шрифта

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    int x, y;
    x = 10;
    y = 10;
```

```
String st = "Microsoft Visual Studio 2010";  
  
// обычный шрифт  
System.Drawing.Font rFont = new System.Drawing.Font  
("Tahoma", 11, FontStyle.Regular);  
  
// полужирный шрифт  
System.Drawing.Font bFont = new System.Drawing.Font  
("Tahoma", 11, FontStyle.Bold);  
  
// курсив  
System.Drawing.Font iFont = new System.Drawing.Font  
("Tahoma", 11, FontStyle.Italic);  
  
// полужирный курсив  
System.Drawing.Font biFont = new System.Drawing.Font  
("Tahoma", 11, FontStyle.Bold | FontStyle.Italic);  
  
// вывод текста различными шрифтами  
e.Graphics.DrawString(st, rFont, Brushes.Black, x, y);  
e.Graphics.DrawString(st, bFont, Brushes.Black, x, y + 20);  
e.Graphics.DrawString(st, iFont, Brushes.Black, x, y + 40);  
e.Graphics.DrawString(st, biFont, Brushes.Black, x, y + 60);  
}
```

Метод `DrawString` позволяет вывести текст в прямоугольную область заданного размера. Если строка длинная, то она автоматически будет разбита на подстроки по границе слов, так что длина каждой подстроки будет не больше ширины области вывода.

Инструкция вызова метода `DrawString`, обеспечивающая вывод текста в область, выглядит так:

```
DrawString(st, aFont, aBrush, aRec);
```

Параметр `aRec` (структура `Rectangle`) задает положение и размер области вывода текста.

В качестве примера в листинге 4.11 приведена функция обработки события `Paint`, которая демонстрирует вывод текста в область. Окно программы приведено на рис. 4.13 (границы области вывода текста показаны для наглядности).

Листинг 4.11. Вывод текста в область

```
private void Form1_Paint(object sender, PaintEventArgs e)  
{  
    String st = "У лукоморья дуб зеленый;,\nЗлатая цепь на дубе том:,\n"+  
    "И днем и ночью кот ученый\nВсе ходят по цепи кругом;";  
  
    // положение и размер области вывода текста  
    Rectangle aRect = new Rectangle(10, 10, 200, 90);
```

```
// вывести текст
e.Graphics.DrawString(st, this.Font, Brushes.Black, aRect);

// показать область отображения текста
e.Graphics.DrawRectangle(Pens.Gray, aRect);
}
```

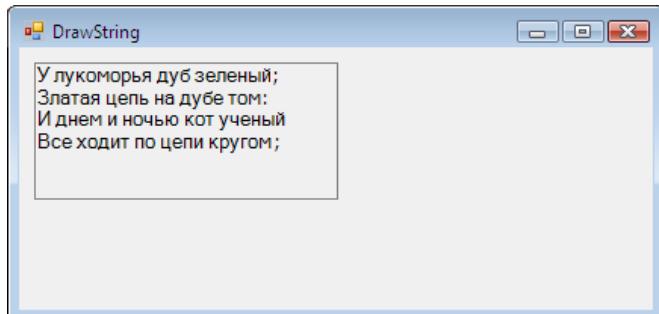


Рис. 4.13. Метод `DrawString` позволяет вывести текст в область

Часто надо знать, сколько места займет текст, например, для того, чтобы правильно разместить его на графической поверхности. Так, чтобы расположить текст по центру формы, надо знать ширину области вывода. Очевидно, что размер области вывода зависит от шрифта, который используется для отображения текста.

Информацию о размере области вывода текста возвращает метод `MeasureString`. В инструкции вызова метода надо указать строку и шрифт, который будет использован для ее отображения. Значением метода `MeasureString` является объект типа `SizeF`, свойства `Width` и `Height` которого содержат информацию о размере области вывода текста.

В листинге 4.12 приведен фрагмент программы, которая демонстрирует использование метода `MeasureString`. Функция обработки события `Paint` выводит текст в центре формы (рис. 4.14). Кроме обработки события `Paint` программа выполняет обработку события `Resize`, которое возникнет, если пользователь изменит размер окна. Обработка заключается в вызове метода `Refresh`, который информирует сис-



Рис. 4.14. Метод `MeasureString` позволяет получить информацию о размере области вывода текста, чтобы расположить текст по центру формы

тему о необходимости обновить (перерисовать) окно, что приводит к возникновению события Paint.

Листинг 4.12. Использование метода MeasureString

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    String st1 = "Microsoft Visual Studio 2010";

    // создать шрифт
    System.Drawing.Font f1 = new System.Drawing.Font("Tahoma", 14);
    System.Drawing.Font f2 = new System.Drawing.Font("Tahoma", 12);
    System.Drawing.Font f3 = new System.Drawing.Font("Tahoma", 8);

    float x, y;
    float w, h; // ширина и высота области, занимаемой текстом

    // определить размер области, которую займет
    // текст при выводе его шрифтом f1
    w = e.Graphics.MeasureString(st1, f1).Width;
    h = e.Graphics.MeasureString(st1, f1).Height;

    // Вычислить координату x, так чтобы текст
    // был размещен в центре окна.
    // ClientSize.Width - ширина внутренней области окна
    x = (this.ClientSize.Width - w)/2;
    y = 10;

    // вывод строки
    e.Graphics.DrawString(st1, f1, Brushes.Black, x, y);

    w = e.Graphics.MeasureString(st1, f2).Width;
    h = e.Graphics.MeasureString(st1, f2).Height;

    x = (this.ClientSize.Width - w)/2;
    y += h;

    e.Graphics.DrawString(st1, f2, Brushes.Black, x, y);

    w = e.Graphics.MeasureString(st1, f3).Width;
    h = e.Graphics.MeasureString(st1, f3).Height;

    x = (this.ClientSize.Width - w)/2;
    y += h;

    e.Graphics.DrawString(st1, f3, Brushes.Black, x, y);
}
```

```
// пользователь изменил размер формы
private void Form1_Resize(object sender, EventArgs e)
{
    // Сообщить системе о необходимости обновить (перерисовать) окно.
    // В результате будет сгенерировано событие Paint.
    this.Refresh();
}
```

Битовые образы

Для формирования сложных изображений используют *битовые образы*. Битовый образ — это небольшая картинка, которая находится в оперативной памяти компьютера. Так как битовый образ находится в оперативной памяти, то его можно очень быстро вывести на экран. Именно поэтому битовые образы используются для формирования картинок в играх.

Создать битовый образ (объект `Bitmap`) можно путем загрузки из файла (`bmp`, `jpg` или `gif`), *ресурса* или путем копирования из другого графического объекта (`Image`).

Загрузку битового образа из файла обеспечивает конструктор, которому в качестве параметра надо передать имя файла. Например, следующий фрагмент кода обеспечивает создание битового образа путем загрузки картинки из файла.

```
System.Drawing.Bitmap sky; // битовый образ
// загрузить из файла
plane = new Bitmap(Application.StartupPath + "\\sky.bmp");
```

Битовый образ можно вывести на графическую поверхность формы или компонента `PictureBox` при помощи метода `DrawImage`. В качестве параметров метода `DrawImage` надо указать битовый образ и координаты точки поверхности, от которой следует вывести битовый образ. Например, инструкция

```
e.Graphics.DrawImage(sky, 0, 0);
```

выводит на графическую поверхность битовый образ `sky`.

Вместо параметров `x` и `y` в инструкции вызова метода `DrawImage` можно указать структуру типа `Point`.

Для битового образа можно задать прозрачный цвет. Точки рисунка, цвет которых совпадает с "прозрачным", при выводе битового образа не отображаются. Прозрачный цвет задает метод `MakeTransparent`. В инструкции вызова метода необходимо указать цвет, который следует рассматривать как прозрачный. Например, инструкция

```
plane.MakeTransparent(System.Drawing.Color.Magenta);
```

задает, что для битового образа `plane` прозрачным является цвет `Magenta` (пурпурный).

В инструкции вызова метода `MakeTransparent` цвет можно не указывать. В этом случае прозрачным будет цвет, которым окрашена левая нижняя точка битового образа.

Программа "Два самолета" (рис. 4.15) демонстрирует загрузку и отображение битовых образов. Небо и самолеты — это битовые образы. Загрузку битовых образов выполняет конструктор формы, вывод — функция события Paint (листинг 4.13). Белое поле вокруг левого самолета показывает реальную форму и размер битового образа plane. Белого поля вокруг правого самолета нет, т. к. перед повторным выводом битового образа plane для него был определен прозрачный цвет.



Рис. 4.15. Задав прозрачный цвет, можно скрыть фон

Листинг 4.13. Загрузка и вывод битовых образов

```
public partial class Form1 : Form
{
    private System.Drawing.Bitmap sky, plane;

    public Form1()
    {
        InitializeComponent();

        // загрузить битовые образы из файла
        try
        {
            sky = new Bitmap(Application.StartupPath + "\\sky.bmp");

            // установить размер формы равным размеру фонового рисунка
            this.ClientSize = sky.Size;

            plane = new Bitmap(Application.StartupPath + "\\plane.bmp");
        }
        catch (System.Exception e)
        {
            ;
        }
    }
}
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    if (sky != null)
    {
        // вывести фоновый рисунок
        e.Graphics.DrawImage(sky, 0, 0);
    }

    if (plane != null)
    {
        // левый самолет
        e.Graphics.DrawImage(plane, 70, 50);

        // правый самолет
        plane.MakeTransparent(); // задать прозрачный цвет

        e.Graphics.DrawImage(plane, 300, 80);
    }
}
```

Анимация

Теперь рассмотрим, как можно использовать битовые образы для создания динамического (меняющегося) изображения — анимации.

Наиболее просто создать эффект меняющейся картинки можно путем вывода на экран (графическую поверхность) последовательности заранее подготовленных картинок (кадров). Кадры анимации обычно помещают в один файл (так удобнее их создавать). В начале работы программы "фильм" загружается в буфер (объект типа `Bitmap`).

Вывести на графическую поверхность фрагмент битового образа (загруженный битовый образ содержит все кадры, а нам нужен отдельный кадр) можно при помощи метода `DrawImage`.

Инструкция вызова метода `DrawImage` для отображения фрагмента битового образа в общем виде выглядит так:

```
Graphics.DrawImage(bitmap, r1, r2, GraphicsUnit.Pixel);
```

где:

- ◆ `Graphics` — поверхность, на которую выполняется вывод фрагмента битового образа `bitmap`;
- ◆ `bitmap` — битовый образ, фрагмент которого выводится на поверхность `Graphics`;
- ◆ `r1` — область на поверхности `Graphics`, в которую выполняется вывод фрагмента битового образа (`объект Rectangle`);

- ◆ `r2` — фрагмент битового образа `bitmap` (объект `Rectangle`), который выводится на поверхность `Graphics`.

Работу метода `DrawImage` при отображении фрагмента битового образа поясняет рис. 4.16.

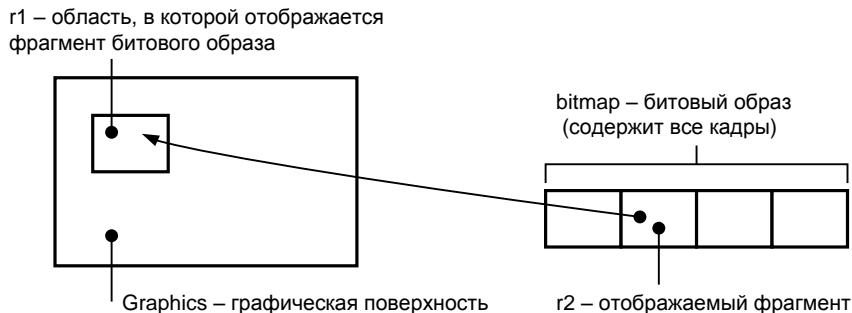


Рис. 4.16. Метод `DrawImage` позволяет вывести на графическую поверхность фрагмент битового образа

Программа *Digital Clock* (рис. 4.17) демонстрирует использование метода `DrawImage` для вывода на графическую поверхность фрагмента битового образа. Цифры, отображаемые в окне, это фрагменты битового образа (рис. 4.18).

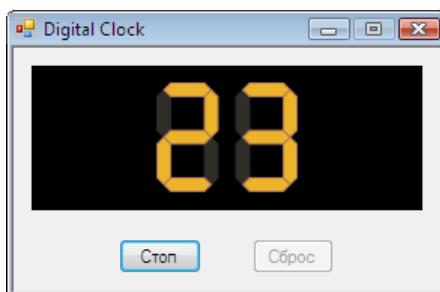


Рис. 4.17. Digital Clock



Рис. 4.18. Битовый образ

Форма программы *Digital Clock* приведена на рис. 4.19, конструктор формы и функции обработки событий — в листинге 4.14.

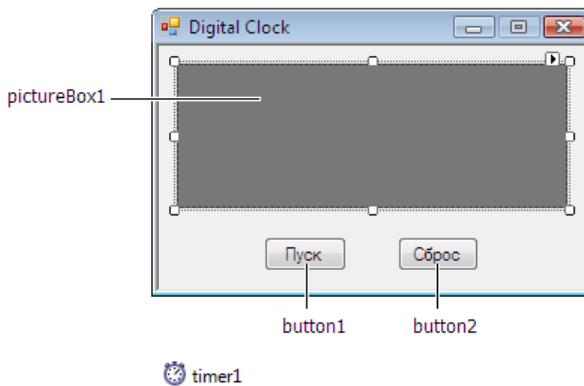


Рис. 4.19. Форма программы Digital Clock

Листинг 4.14. Digital Clock

```

public partial class Form1 : Form
{
    Bitmap digits; // цифры

    // размер кадра (цифры)
    int wd;
    int hd;
    int x0,y0; // левый верхний угол индикатора
    int s;      // счетчик секунд

    public Form1()
    {
        InitializeComponent();

        digits = new Bitmap(Application.StartupPath + "\\digits_3.bmp");

        // задать цвет фона pictureBox1, совпадающий с цветом
        pictureBox1.BackColor = digits.GetPixel(1, 1);

        // размер битового образа цифры
        hd = digits.Height;
        wd = (digits.Width)/10;

        // положение левого верхнего угла индикатора (2 цифры)
        // в поле pictureBox
        x0 = (pictureBox1.Width - 2*wd)/2;
        y0 = (pictureBox1.Height - hd) /2;

        s = 0;

        timer1.Interval = 1000;
    }
}

```

```
// щелчок на кнопке Пуск/Стоп
private void button1_Click(object sender, EventArgs e)
{
    timer1.Enabled = !timer1.Enabled;
    if (timer1.Enabled)
    {
        button1.Text = "Стоп";
        button2.Enabled = false;
    }
    else
    {
        button1.Text = "Пуск";
        button2.Enabled = true;
    }
}

// сигнал от таймера
private void timer1_Tick(object sender, EventArgs e)
{
    if (s < 59) s++;
    else s = 0;

    // перерисовать компонент pictureBox1
    pictureBox1.Refresh();
}

private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    int d0, d1;

    d0 = s % 10; // кол-во единиц секунд (младший разряд)
    d1 = s / 10; // кол-во десятков секунд (старший разряд)

    Rectangle r1 = new Rectangle(); // куда копировать фрагмент
                                    // битового образа
    Rectangle r2 = new Rectangle(); // откуда (положение и размер
                                    // копируемого фрагмента)

    // старший разряд (десятки секунд):
    // куда копировать
    r1.X = x0;
    r1.Y = y0;
    r1.Width = wd;
    r1.Height = hd;

    // задать положение фрагмента, в котором находится нужная цифра
    r2 = new Rectangle(x0, y0, wd, hd);
```

```
r2.X = d1 * wd;
r2.Y = 0;
r2.Width = wd;
r2.Height = hd;

// копировать фрагмент битового образа digits
// на поверхность компонента picturePox1
e.Graphics.DrawImage(digits, r1, r2, GraphicsUnit.Pixel);

// секунды (младший разряд):
r1.X = r1.X + wd;      // куда копировать
r2.X = d0 * wd;         // откуда копировать

e.Graphics.DrawImage(digits, r1, r2, GraphicsUnit.Pixel);
}

// сигнал на кнопке Сброс
private void button2_Click(object sender, EventArgs e)
{
    s = 0;
    pictureBox1.Refresh();
}
}
```

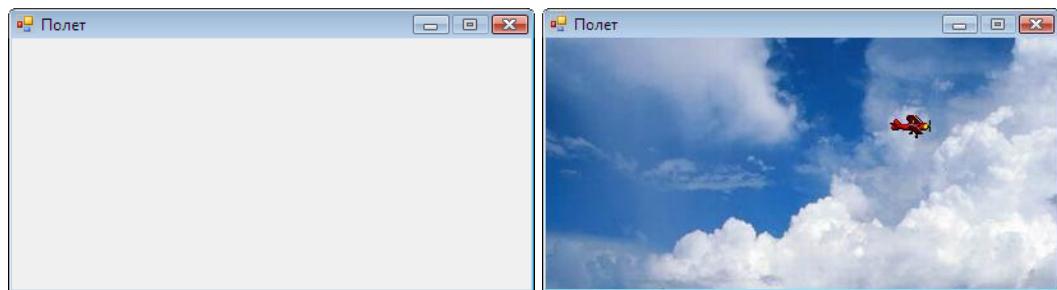
Конструктор загружает битовый образ из файла, и, используя информацию о размере загруженного битового образа, функция устанавливает значения характеристик кадра: высоту и ширину (в файле находится изображение 10 цифр, поэтому ширина кадра равна 1/10 ширины загруженного битового образа). Основную работу в программе выполняет функция обработки события Paint компонента PictureBox. Она определяет цифры, которые надо вывести, и выводит их на графическую поверхность компонента. Положение фрагмента битового образа (координата x левого верхнего угла) для старшего разряда индикатора определяется умножением текущего значения d0 (количество десятков секунд) на ширину кадра. Аналогичным образом определяется положение фрагмента битового образа для младшего разряда индикатора. Событие Paint компонента PictureBox возникает с периодом, равным периоду возникновения события Timer — таймер путем вызова метода Refresh инициирует возникновение события Paint компонента PictureBox.

В программе Digital Clock для формирования динамической картинки использовался "классический" подход создания анимации, предполагающий наличие заранее подготовленной серии картинок (кадров), последовательное отображение которых и создает эффект анимации. Возможен и другой подход к созданию динамических изображений, когда картинка (кадр) формируется из заранее подготовленных фрагментов "на лету", во время работы программы.

Типичным примером такой анимации является перемещение объекта на фоне какой-либо картинки. Чтобы у наблюдателя сложилось впечатление, что объект

двигается, надо вывести изображение объекта, затем, через некоторое время, стереть его и снова вывести, но уже на некотором расстоянии от первоначального положения. Подбором времени между удалением и выводом изображения, а также расстояния между новым и предыдущим положением объекта (шага перемещения), можно добиться эффекта равномерного движения.

Программа "Полет" (ее форма и окно приведены на рис. 4.20, а текст — в листинге 4.15) демонстрирует принципы создания анимации "на лету", показывает, как заставить объект двигаться.



timer1

Рис. 4.20. Форма и окно программы "Полет"

Листинг 4.15. Движение объекта

```
public partial class Form1 : Form
{
    // битовые образы: небо и самолет
    System.Drawing.Bitmap sky, plane;

    Graphics g; // рабочая графическая поверхность

    // приращение координаты X, определяет скорость полета
    int dx;

    // область, в которой находится самолет
    Rectangle rct;

    // true - самолет скрывается в облаках
    Boolean demo = true;

    // генератор случайных чисел
    System.Random rnd;

    public Form1()
    {
        InitializeComponent();
    }
```

```
try
{
    // Вариант 1: загрузка битовых образов из файлов
    sky = new Bitmap("sky.bmp");           // небо
    plane = new Bitmap("plane.bmp");        // самолет

    // загрузить и задать фоновый рисунок формы
    this.BackgroundImage = new Bitmap("sky.bmp");

    // Вариант 2: загрузка битовых образов из ресурса
    /*
    sky = Properties.Resources.sky;         // фон
    plane = Properties.Resources.plane;     // самолет

    // загрузить и задать фоновый рисунок формы
    this.BackgroundImage = Properties.Resources.sky;
    */

}
catch (Exception exception)
{
    MessageBox.Show("Ошибка доступа к файлам битовых образов.",
                    "Полет",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

// сделать прозрачным фон вокруг объекта
plane.MakeTransparent();

// задать размер формы в соответствии
// с размером фонового рисунка
this.ClientSize = new System.Drawing.Size(
    new Point(BackgroundImage.Width,
              BackgroundImage.Height));

// задать вид границы окна
this.FormBorderStyle =
    System.Windows.Forms.FormBorderStyle.FixedSingle;
this.MaximizeBox = false;

// g - графическая поверхность, на которой
// будем формировать рисунок.
// В качестве поверхности будем использовать BackgroundImage формы
g = Graphics.FromImage(BackgroundImage);

// инициализация генератора случайных чисел
rnd = new System.Random();
```

```
// исходное положение самолета
rct.X = -40;
rct.Y = 20 + rnd.Next(20);

rct.Width = plane.Width;
rct.Height = plane.Height;

/*
скорость полета определяется периодом следования
сигналов от таймера (значение свойства Timer1.Interval)
и величиной приращения координаты по X
*/
dx = 2;           // скорость полета - 2 пиксела/тик_таймера

timer2.Interval = 20;
timer2.Enabled = true;
}

private void timer2_Tick(object sender, EventArgs e)
{
    // стираем изображение самолета путем копирования
    // области фона на рабочую поверхность
    g.DrawImage(sky,new Point(0,0));

    // изменяем положение самолета
    if (rct.X < this.ClientRectangle.Width)
        rct.X += dx;
    else {
        // если достигли границы, задаем заново положение самолета
        rct.X = -40;
        rct.Y = 20 + rnd.Next(this.ClientSize.Height - 40 - plane.Height);

        // скорость полета от 2 до 5 пикселов/тик_таймера
        dx = 2 + rnd.Next(4);
    }

    // рисуем самолет на рабочей поверхности (фактически
    // на поверхности формы), но для того чтобы изменения появились,
    // надо иницировать обновление формы
    g.DrawImage(plane, rct.X, rct.Y);

    /*
    Метод Refresh инициирует перерисовку всей формы.
    Метод Invalidate позволяет иницировать перерисовку
    только той области формы, которая указана
    в качестве параметра метода.
    */
}
```

```
if (! demo)
    // обновить область формы, в которой находится объект
    this.Invalidate(rct);
else
{
    // если объект находится вне области, указанной в качестве
    // параметра метода Invalidate, то от нее будет виден
    Rectangle reg = new Rectangle(20, 20,
                                   sky.Width - 40, sky.Height - 40);

    // показать обновляемую область
    g.DrawRectangle(Pens.Black,
                    reg.X, reg.Y, reg.Width-1, reg.Height-1);

    this.Invalidate(reg); // обновить область
}
}
```

}

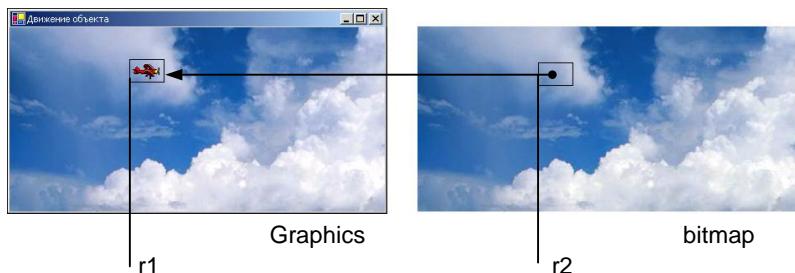
}

В рассматриваемой программе фоновый рисунок и изображение объекта загружаются из файлов. Конструктор формы обеспечивает загрузку битовых образов, задает начальное положение объекта, выполняет настройку и запуск таймера. Следует обратить внимание на то, что начальное значение поля *x* структуры *rec*, которое определяет положение левого верхнего угла битового образа (движущейся картинки), — отрицательное число, равное ширине битового образа. Поэтому в начале работы программы самолет не виден, картинка отрисовывается за границей видимой области. С каждым событием *Tick* значение координаты *x* увеличивается, и на экране появляется та часть битового образа, координаты которой больше нуля. Таким образом, у наблюдателя создается впечатление, что самолет вылетает из-за левой границы окна.

Процедура обработки сигнала от таймера (события *Timer*) выполняет основную работу. Сначала она стирает изображение объекта (восстанавливает "испорченный" фоновый рисунок), затем выводит изображение объекта на новом месте. Восстановление фона выполняет метод *DrawImage* путем вывода фрагмента фонового рисунка (битового образа *sky*) в область графической поверхности, в которой находится в данный момент объект (рис. 4.21).

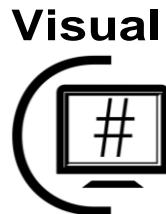
Следует обратить внимание на то, что в функции обработки события *Timer* для доступа к графической поверхности используется переменная *g*. Эта переменная хранит ссылку на графическую поверхность окна программы. Ее значение устанавливает конструктор формы: метод *CreateGraphics* возвращает ссылку на графическую поверхность объекта *this*, т. е. формы. Такой "хитрый" способ используется потому, что доступ к графической поверхности (через параметр *e*) есть у функции обработки события *Paint* и, в принципе, выводить графику должна процедура обработки именно этого события. В нашем примере картинка обновляется несколько

раз в секунду и перерисовка окна (а именно это делает функция обработки события Paint) привела бы к заметному мерцанию картинки.



`Graphics.DrawImage(bitmap, r1, r2, GraphicsUnit.Pixel)`

Рис. 4.21. Метод `DrawImage` позволяет вывести в указанную область графической поверхности фрагмент битового образа



ГЛАВА 5

Базы данных

Microsoft Visual C# предоставляет программисту набор компонентов, используя которые он может создать программу работы практически с любой базой данных: от Microsoft Access до Microsoft SQL Server и Oracle.

База данных и СУБД

Физически база данных — это файл или совокупность файлов определенной структуры, в которых находится информация; логически база данных — это, если речь идет о реляционной базе данных, совокупность связанных таблиц, в которых находится информация. Программная система, обеспечивающая работу с базой данных, называется *системой управления базой данных (СУБД)*. СУБД позволяет создать базу данных, наполнить ее информацией, решить задачи просмотра (отображения), поиска, архивирования и др. Типичным примером СУБД является Microsoft Access.

Локальные и удаленные базы данных

В зависимости от расположения данных и приложения, обеспечивающего работу (доступ) с ними, различают *локальные* и *удаленные* базы данных.

В локальной базе данных файлы данных, как правило, находятся на диске того компьютера, на котором работает программа манипулирования данными. Локальные базы данных не обеспечивают одновременный доступ к информации нескольким пользователям. Несомненным достоинством локальной базы данных является высокая скорость доступа к информации. Microsoft Access — это типичная локальная база данных.

В удаленных базах данных файлы данных размещают на отдельном, доступном по сети, компьютере (сервере). Программы, обеспечивающие работу с удаленными базами данных, строят по технологии "клиент-сервер". Программа-клиент, работающая на компьютере пользователя, обеспечивает доступ к данным (прием команд от пользователя, передачу их серверу, получение и отображение данных). Серверная часть (сервер), работающая на удаленном компьютере, принимает за-

просы (команды) от клиента, выполняет их и пересыпает данные клиенту. Программа, работающая на удаленном компьютере, проектируется так, чтобы обеспечить одновременный доступ к базе данных многим пользователям. В большинстве случаев в качестве серверной части используется стандартный сервер баз данных, например Microsoft SQL Server. Таким образом, разработка программы работы с удаленной базой данных в большинстве случаев сводится к разработке программы-клиента.

Структура базы данных

База данных (в широком смысле) — это набор однородной и, как правило, упорядоченной по некоторому критерию информации.

На практике наиболее широко используются *реляционные* базы данных (от англ. *relation* — отношение, таблица). Реляционная база данных — это совокупность связанных таблиц данных. Так, например, базу данных Projects (Проекты) можно представить как совокупность таблиц Projects (Проекты), Tasks (Задачи) и Resources (Ресурсы), а базу данных Contacts (Контакты) — одной-единственной таблицей Contacts (Контакты). Доступ к таблице осуществляется по имени.

Строки таблиц данных называются *записями*. Они содержат информацию об объектах базы данных. Например, строка таблицы Tasks (Задачи) базы данных Projects (Проекты) может содержать название задачи, дату, когда должна быть начата работа, и идентификатор ресурса, который назначен на выполнение задачи. Доступ к записям осуществляется по номеру.

Записи состоят из полей (поле — ячейка в строке таблицы). Поля содержат информацию о характеристиках объекта. Доступ к полю осуществляется по имени. Например, поля записей таблицы Tasks могут содержать: идентификатор задачи (поле TaskID), название задачи (поле title), идентификатор проекта, частью которого является задача (поле ProjID), дату, когда работа по выполнению задачи должна быть начата (поле Start), информацию о состоянии задачи (поле Status) и идентификатор ресурса, который назначен на выполнение задачи (поле ResID). При представлении данных в табличной форме имена полей указывают в заголовке (в первой строке) таблицы.

Физически база данных представляет собой файл или совокупность файлов, в которых находятся таблицы. Например, в Microsoft Access все таблицы, образующие базу данных, хранятся в одном файле с расширением mdb.

Компоненты доступа к данным

Для доступа к данным в .NET-приложениях Microsoft рекомендует использовать технологию ADO.NET — универсальный механизм доступа к базам данных. Ее несомненное достоинство — возможность доступа к различным источникам данных. Поддержку ADO.NET обеспечивают компоненты OleDbConnection и OleDbDataAdapter (рис. 5.1).

Если компоненты и OleDbDataAdapter в палитре компонентов не отображаются, то для того чтобы они стали доступны, сделайте щелчок правой кнопкой мыши в

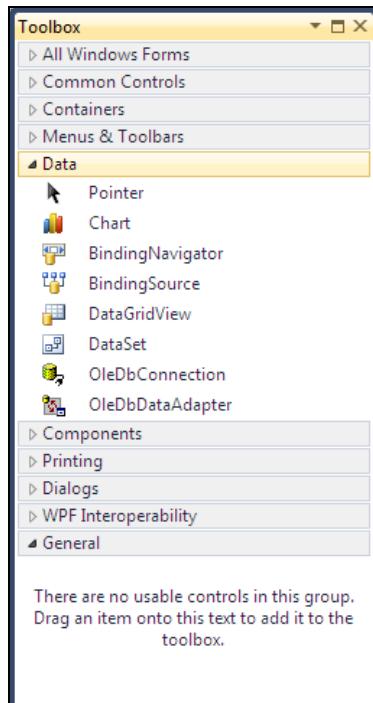


Рис. 5.1. Компоненты OleDbConnection и OleDbDataAdapter

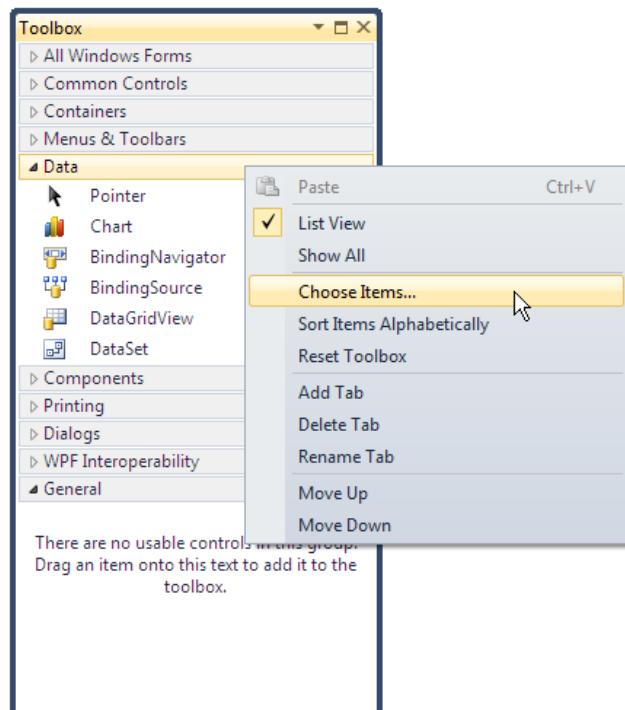


Рис. 5.2. Чтобы добавить в палитру один или несколько компонентов, выберите команду **Choose Items...**

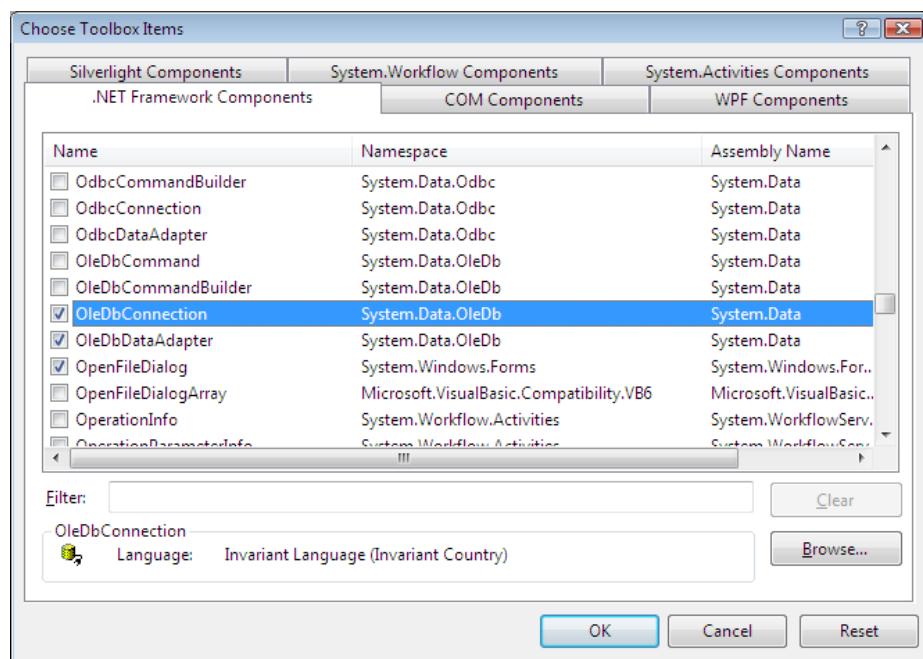


Рис. 5.3. Подключение компонентов OleDbConnection и OleDbDataAdapter

заголовке вкладки **Data** палитры компонентов и в появившемся контекстном меню выберите команду **Choose Items** (рис. 5.2). Затем в открывшемся окне **Choose Toolbox Items** раскройте вкладку **.NET Framework Components**, найдите в списке компоненты `OleDbConnection` и `OleDbDataAdapter` и установите соответствующие им флажки (рис. 5.3).

Создание базы данных

Программы работы с базами данных обычно оперируют с существующими файлами данных и, как правило, не предоставляют пользователю возможность создать базу данных. Поэтому, перед тем как приступить к разработке приложения работы с базой данных, необходимо с помощью соответствующей СУБД создать базу данных.

База данных Microsoft Access

Процесс разработки программы работы с базой данных Microsoft Access рассмотрим на примере. Создадим программу, обеспечивающую работу с базой данных "Контакты".

Перед тем как приступить непосредственно к работе над программой, необходимо с помощью Microsoft Access создать базу данных "Контакты" (файл `contacts.mdb`), содержащую таблицу `contacts` (табл. 5.1). Файл базы данных следует поместить, например, в папку Документы\Contacts (папку Contacts надо создать в папке Документы). Кроме этого, в папке Contacts надо создать папку Images (в ней будем хранить иллюстрации — фотографии "контактов").

Таблица 5.1. Таблица contacts базы данных "Контакты"

Поле	Тип	Размер	Описание
<code>cid</code>	Целое, автоувеличение	—	Уникальный идентификатор контакта
<code>name</code>	Текстовый	50	Имя
<code>phone</code>	Текстовый	50	Телефон
<code>email</code>	Текстовый	50	Адрес электронной почты
<code>img</code>	Текстовый	50	Файл иллюстрации

Надо обратить внимание на следующее. Microsoft Access — это СУБД, *приложение*, обеспечивающее работу с базами данных. Для работы с данным Microsoft Access использует так называемое ядро баз данных Microsoft Jet.

Доступ к данным

Доступ к базе данных обеспечивают компоненты `OleDbConnection`, `OleDbDataAdapter` и `DataSet`.

Компонент `OleDbConnection` обеспечивает соединение с базой данных (источником данных), компонент `OleDbDataAdapter` — взаимодействие с базой данных, а `DataSet` — хранение данных, полученных от источника данных в результате выполнения SQL-запроса.

Отображение данных в табличной форме обеспечивает компонент `DataGridView`.

Механизм взаимодействия компонентов, обеспечивающих доступ к данным и их отображение, показан на рис. 5.4.

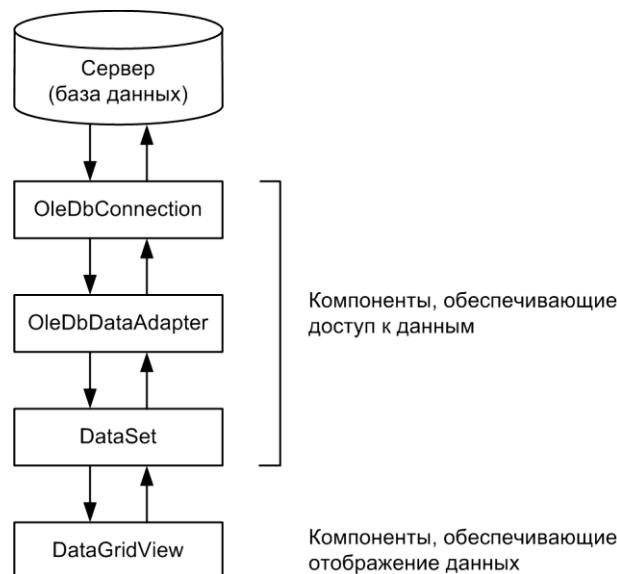
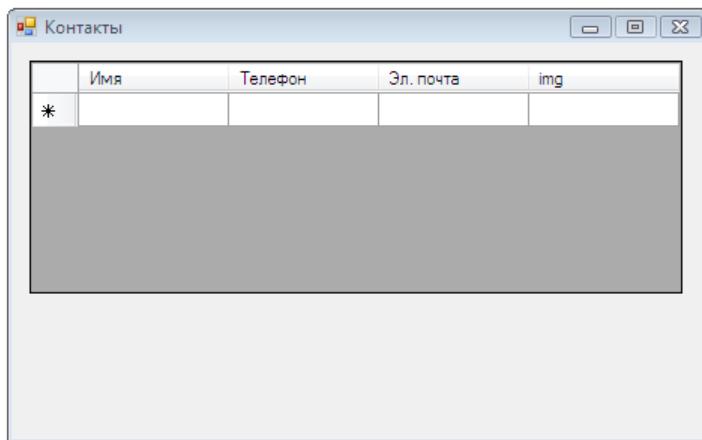


Рис. 5.4. Взаимодействие компонентов, обеспечивающих доступ к данным и их отображение

Форма программы работы с базой данных "Контакты" приведена на рис. 5.5. Сначала на форму надо поместить компонент `OleDbConnection`, затем — `OleDbDataAdapter`, `DataSet` и `DataGridView`. Компоненты рекомендуется добавлять и настраивать в том порядке, в котором они перечислены (см. далее).

Компонент `OleDbConnection` настраивается следующим образом. Сначала надо сделать щелчок на значке раскрывающегося списка, который находится в строке свойства `ConnectionString` (строка соединения), и выбрать **New Connection** (Новое соединение). Затем в появившемся окне **Add Connection** надо сделать щелчок на кнопке **Change** и в появившемся окне **Change Data Source** (рис. 5.6) выбрать тип источника данных (в нашем случае — Microsoft Access Database File). Далее, во вновь ставшем доступном окне **Add Connection** надо сделать щелчок на кнопке **Browse** и указать файл базы данных (в нашем случае — contacts.mdb). Окно создания соединения после выполнения описанных действий должно выглядеть так, как показано на рис. 5.7. После этого можно сделать щелчок на кнопке **Test Connection** и убедиться, что соединение с базой данных настроено правильно.

Завершив настройку соединения с базой данных (компонентом `OleDbConnection`), можно приступить к настройке компонента `OleDbDataAdapter`.



oleDbConnection1 oleDbDataAdapter1 dataSet1

Рис. 5.5. Форма программы работы с базой данных "Контакты"

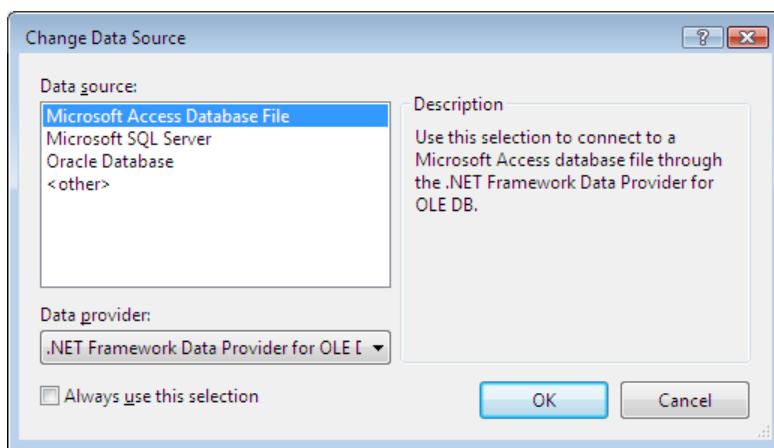


Рис. 5.6. Настройка соединения с базой данных (шаг 1)

Компонент OleDbDataAdapter обеспечивает взаимодействие с базой данных. Его свойства содержат SQL-команды выборки (`SELECT`), добавления (`INSERT`), изменения (`UPDATE`) и удаления (`DELETE`) данных. Свойства компонента OleDbDataAdapter приведены в табл. 5.2.

Таблица 5.2. Свойства компонента `OleDbDataAdapter`

Свойство	Описание
SelectCommand	SQL-команда <code>SELECT</code> , обеспечивающая выбор информации из базы данных
DeleteCommand	SQL-команда <code>DELETE</code> , обеспечивающая удаление информации из базы данных

Таблица 5.2 (окончание)

Свойство	Описание
InsertCommand	SQL-команда INSERT, обеспечивающая добавление информации в базу данных
UpdateCommand	SQL-команда UPDATE, обеспечивающая обновление информации в базе данных

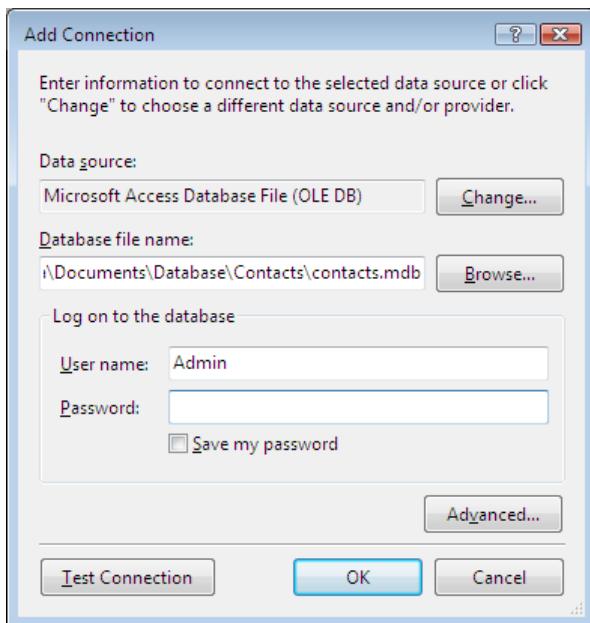


Рис. 5.7. Настройка соединения с базой данных (шаг 2)

Свойства SelectCommand, DeleteCommand, InsertCommand и UpdateCommand представляют собой объекты OleDbCommand, свойства которых (табл. 5.3) определяют соответствующие SQL-команды.

Таблица 5.3. Свойства объекта OleDbCommand

Свойство	Описание
Connection	Объект OleDbConnection, обеспечивающий соединение с базой данных
CommandText	SQL-команда
Parameters	Параметры команды. Коллекция объектов OleDbParameter

Выполнить настройку компонента OleDbDataAdapter можно при помощи мастера настройки или вручную.

Мастер настройки запускается автоматически в результате добавления компонента на форму (чтобы запустить мастер настройки компонента OleDbDataAdapter

вручную, надо выбрать компонент и сделать щелчок на находящейся в нижней части окна **Properties** ссылке **Configure Data Adapter**). В первом окне мастера (рис. 5.8) надо выбрать соединение (connection) и нажать кнопку **Next**. В следующем окне (рис. 5.9) также надо нажать кнопку **Next**.

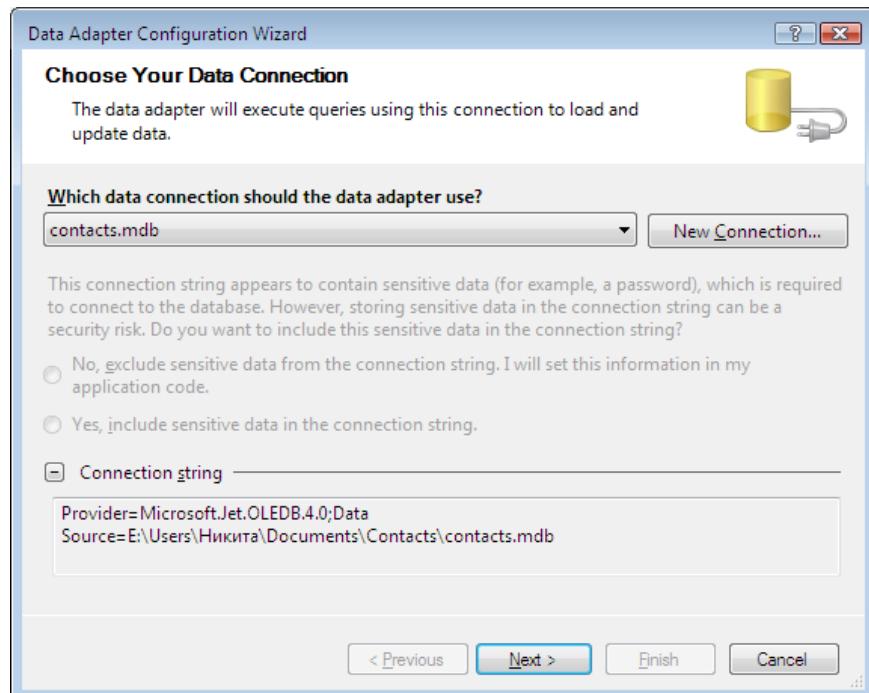


Рис. 5.8. Мастер настройки компонента OleDbDataAdapter (шаг 1)

В окне (рис. 5.10), которое появляется на третьем шаге, надо нажать кнопку **Query Builder**, а затем в окне **Add Table** (рис. 5.11) выбрать таблицу данных и нажать кнопку **Add**.

Далее в окне **Query Builder** (рис. 5.12) следует указать поля таблицы, из которых надо выбрать данные. Если необходимо получить данные из всех полей — выберите **All Columns**.

В следующем окне (рис. 5.13), которое появляется в результате нажатия кнопки **OK**, надо сделать щелчок на кнопке **Finish**.

В результате выполнения описанных действий компонент OleDbDataAdapter будет настроен — в свойства SelectCommand, DeleteCommand, InsertCommand и UpdateCommand будут записаны команды, обеспечивающие взаимодействие с базой данных.

При "ручной" настройке компонента OleDbDataAdapter в первом окне мастера настройки надо нажать кнопку **Cancel** и затем установить значения свойств компонента. В свойства SelectCommand, DeleteCommand, InsertCommand и UpdateCommand надо ввести соответствующие команды. Если команда предполагает использование

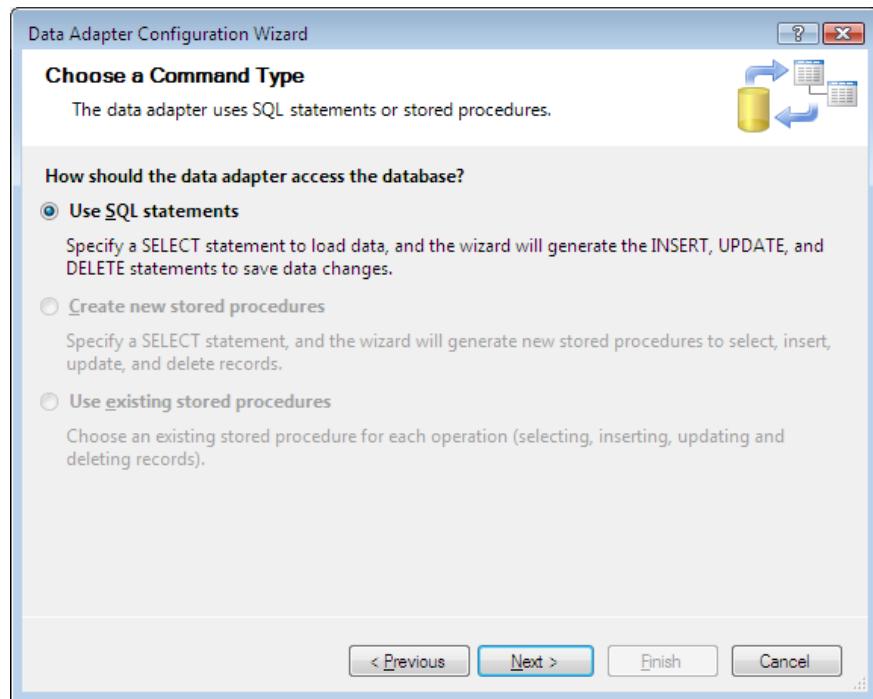


Рис. 5.9. Мастер настройки компонента OleDbDataAdapter (шаг 2)

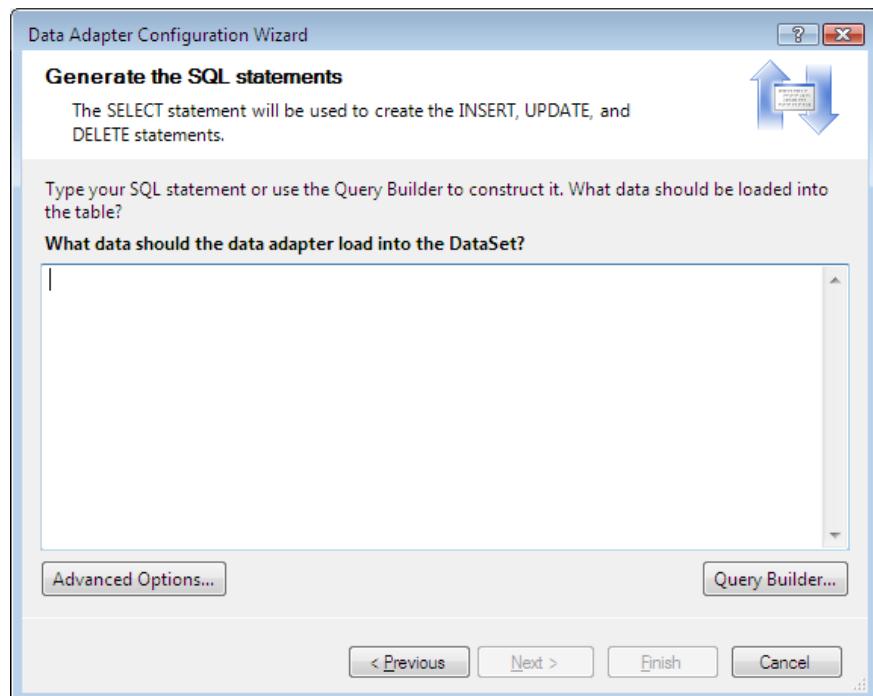


Рис. 5.10. Мастер настройки компонента OleDbDataAdapter (шаг 3)

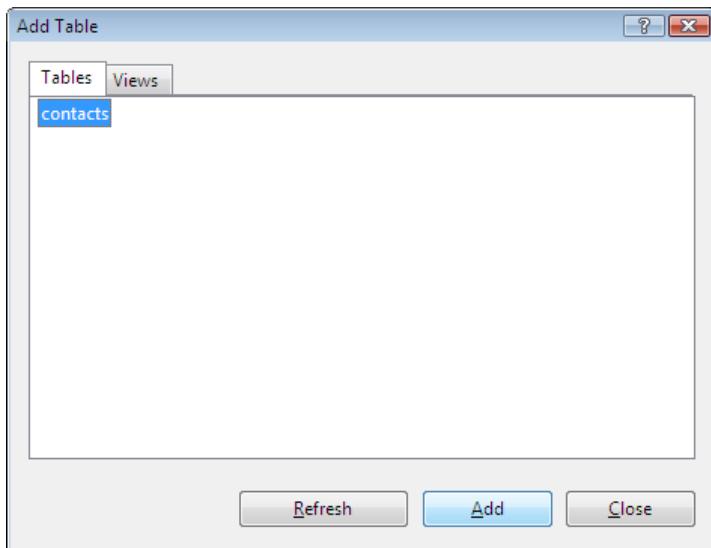


Рис. 5.11. Выбор таблицы данных

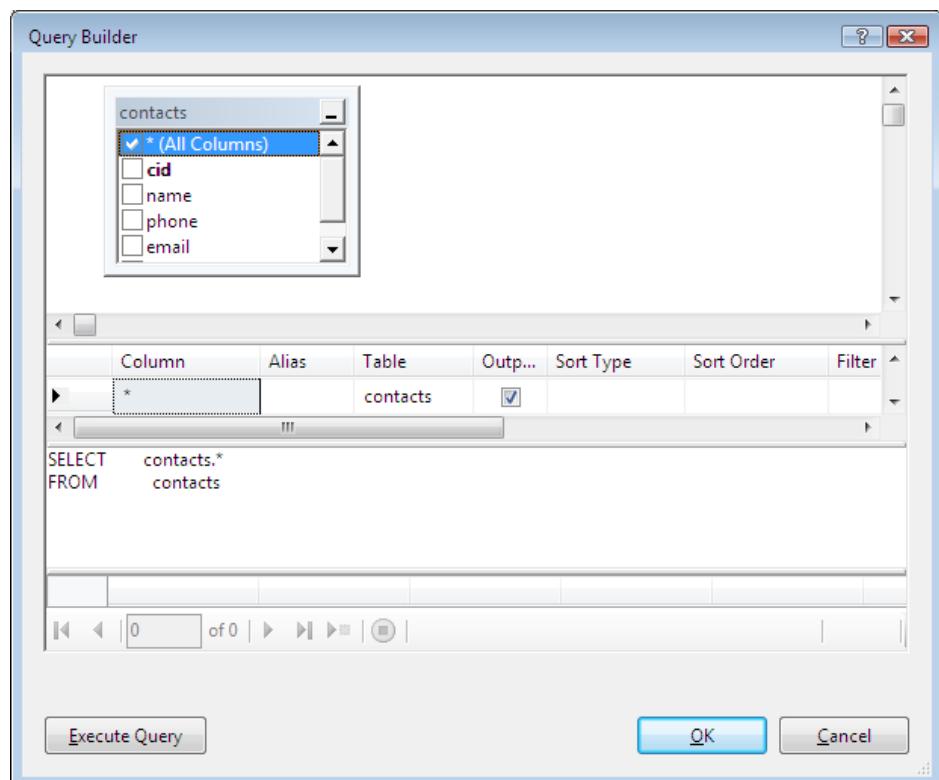


Рис. 5.12. Построение запроса к БД

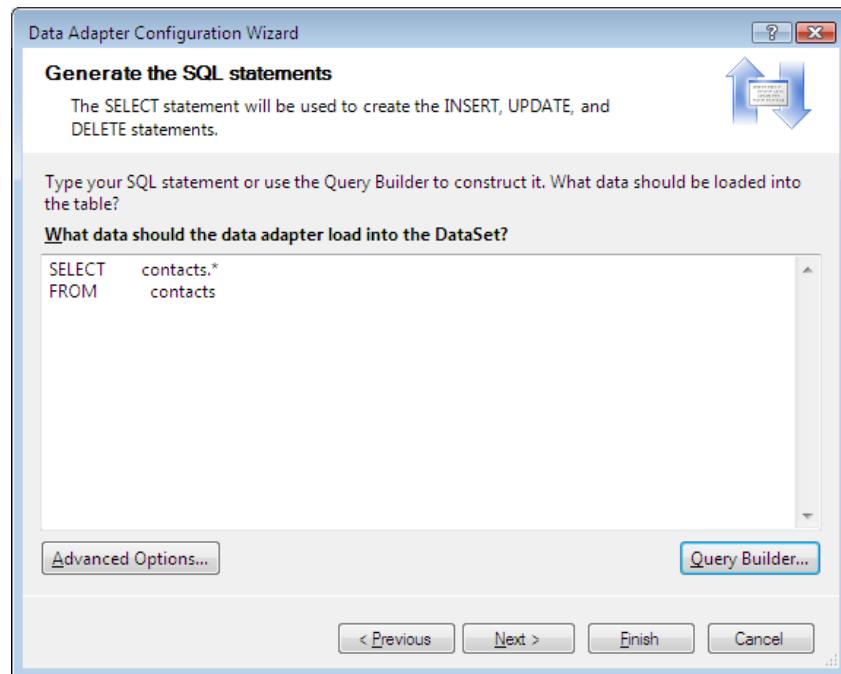


Рис. 5.13. Чтобы завершить настройку соединения, нажмите кнопку **Finish**

параметров, то в позиции параметра указывается символ "?". Например, команда добавления информации в таблицу contacts выглядит так:

```
INSERT INTO contacts(name, phone, email, img) VALUES (?, ?, ?, ?)
```

Если у команды есть параметры, то в коллекцию **Parameters** этой команды надо добавить соответствующее количество элементов. Для этого в окне **Properties** в строке значения свойства **Parameters** надо сделать щелчок на кнопке с тремя точками (рис. 5.14), в появившемся окне **OleDbParameter Collection Editor** нужное количество раз нажать кнопку **Add** и задать значения свойств параметров (рис. 5.15).

Результат "ручной" настройки компонента **OleDbDataAdapter** приведен в табл. 5.4.

Таблица 5.4. Значения свойств компонента **OleDbDataAdapter**

Свойство	Значение
SelectCommand.Connection	oleDbConnection1
SelectCommand.CommandText	SELECT * FROM contacts
InsertCommand.Connection	oleDbConnection1
InsertCommand.CommandText	INSERT INTO contacts (name, phone, email, img) VALUES (?, ?, ?, ?)
InsertCommand.Parameters[0].ParameterName	name

Таблица 5.4 (окончание)

Свойство	Значение
InsertCommand.Parameters[0].SourceColumn	name
InsertCommand.Parameters[1].ParameterName	phone
InsertCommand.Parameters[1].SourceColumn	phone
InsertCommand.Parameters[2].ParameterName	email
InsertCommand.Parameters[2].SourceColumn	email
InsertCommand.Parameters[3].ParameterName	img
InsertCommand.Parameters[3].SourceColumn	img
UpdateCommand.Connection	oleDbConnection1
UpdateCommand.CommandText	UPDATE contacts SET name = ?, phone = ?, email = ?, img = ? WHERE (cid = ?)
UpdateCommand.Parameters[0].ParameterName	name
UpdateCommand.Parameters[0].SourceColumn	name
UpdateCommand.Parameters[1].ParameterName	phone
UpdateCommand.Parameters[1].SourceColumn	phone
UpdateCommand.Parameters[2].ParameterName	email
UpdateCommand.Parameters[2].SourceColumn	email
UpdateCommand.Parameters[3].ParameterName	img
UpdateCommand.Parameters[3].SourceColumn	img
UpdateCommand.Parameters[4].ParameterName	Original_cid
UpdateCommand.Parameters[4].SourceColumn	cid
UpdateCommand.Parameters[4].SourceVersion	Original
DeleteCommand.Connection	oleDbConnection1
DeleteCommand.CommandText	DELETE FROM contacts WHERE (cid = ?)
DeleteCommand.Parameters[0].ParameterName	cid
DeleteCommand.Parameters[0].SourceColumn	cid
DeleteCommand.Parameters[0].SourceVersion	Original

Компонент **DataSet** (набор данных) хранит данные, полученные из базы данных. Свойства компонента **DataSet** приведены в табл. 5.5.

Элемент коллекции **Tables** представляет собой объект **DataTable** (табл. 5.6). Именно он в конечном итоге хранит таблицы — данные, полученные из БД.

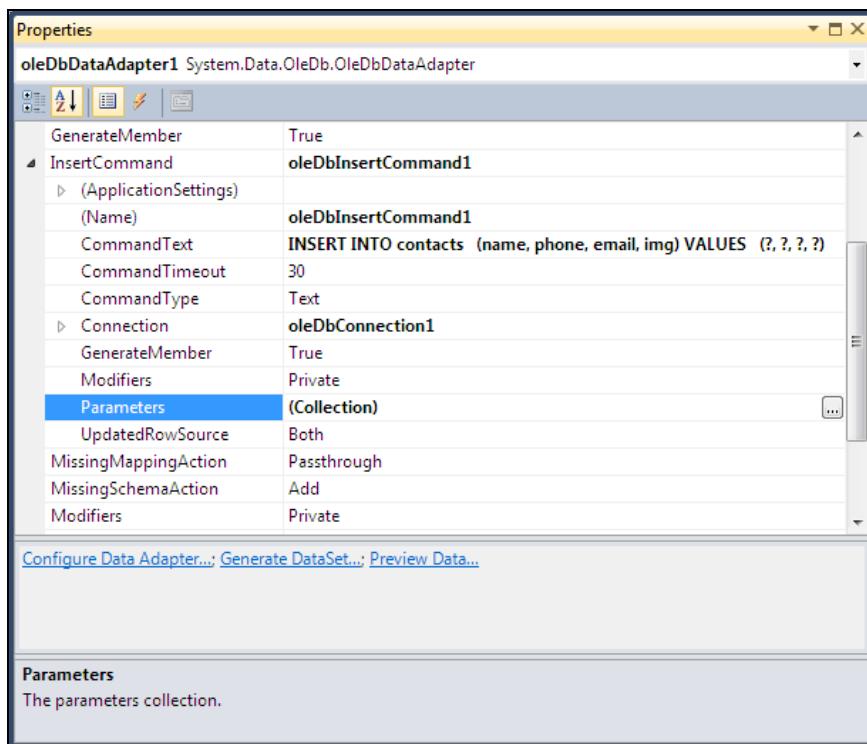


Рис. 5.14. Чтобы добавить параметры, нажмите кнопку с тремя точками

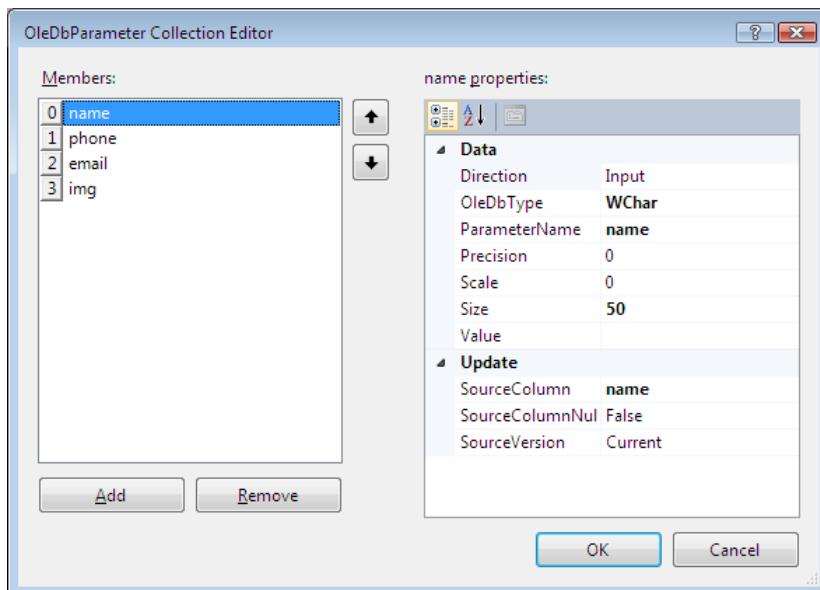


Рис. 5.15. Настройка параметров команды

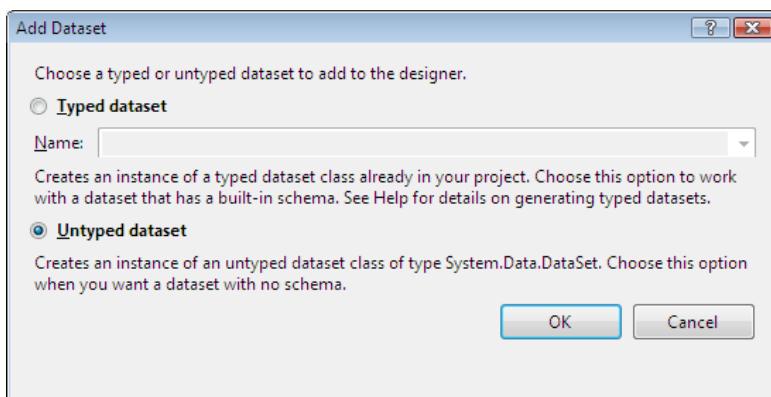
Таблица 5.5. Свойства компонента *DataSet*

Свойство	Описание
DataSetName	Имя набора данных
Tables	Данные, загруженные из базы данных. Коллекция объектов <i>DataTable</i>

Таблица 5.6. Свойства объекта *DataTable*

Свойство	Определяет
Name	Имя таблицы. Используется для доступа к таблице (элементу коллекции <i>Tables</i> объекта <i>DataSet</i>)
Columns	Столбцы таблицы

Настройка компонента *DataSet* выполняется следующим образом. Сначала в окне **Add Dataset** (рис. 5.15), которое появляется на экране в момент добавления компонента на форму, надо выбрать переключатель **Untyped dataset** и нажать кнопку **OK**.

**Рис. 5.16.** Настройка компонента *DataSet* (шаг 1)

Далее, в коллекцию *Tables* надо добавить таблицу (рис. 5.17), а в коллекцию *Columns* — столбцы и у каждого элемента коллекции *Columns* (рис. 5.18) установить значение свойства *ColumnName*. Результат настройки компонента *dataSet1* приведен в табл. 5.7.

Таблица 5.7. Значения свойств компонента *dataSet1*

Свойство	Значение
Name	dataSet1
Tables[0].TableName	contacts
Tables[0].Columns[0].ColumnName	cid
Tables[0].Columns[0].AutoIncrement	true
Tables[0].Columns[1].ColumnName	name

Таблица 5.7 (окончание)

Свойство	Значение
Tables[0].Columns[2].ColumnName	phpone
Tables[0].Columns[3].ColumnName	email
Tables[0].Columns[4].ColumnName	img

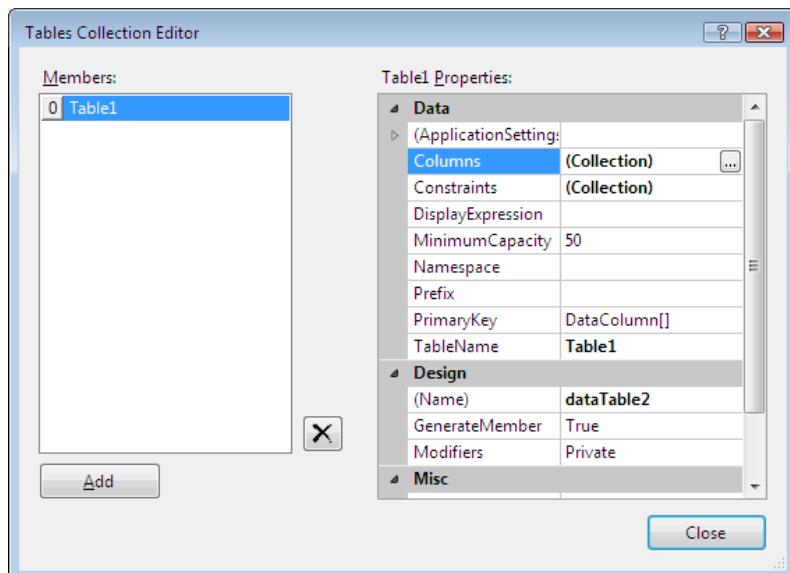


Рис. 5.17. Коллекция Tables

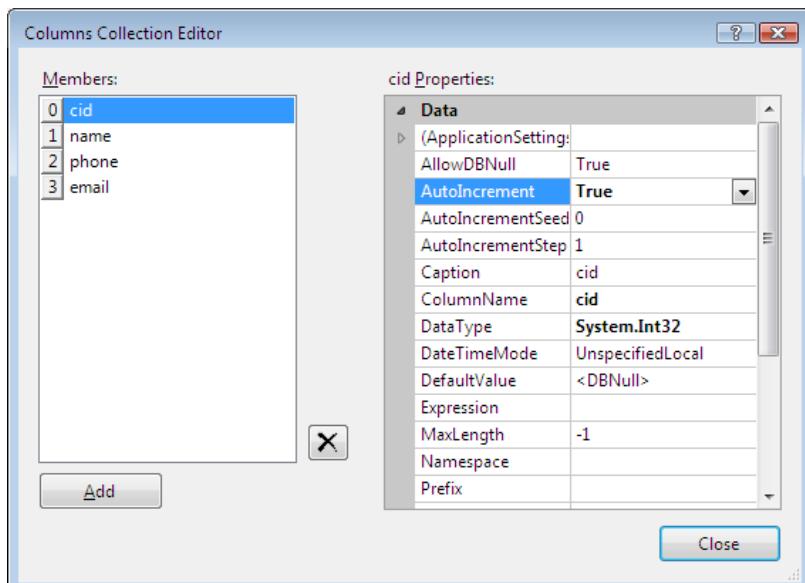


Рис. 5.18. Коллекция Columns

Отображение данных

Пользователь может работать с базой данных в режиме таблицы или в режиме формы. В режиме таблицы информация отображается в виде таблицы, что позволяет видеть одновременно несколько записей. Этот режим обычно используется для просмотра информации. В режиме формы отображается одна запись. Обычно данный режим используется для ввода и редактирования информации. Часто эти два режима комбинируют. Краткая информация (содержимое некоторых ключевых полей) выводится в табличной форме, а при необходимости видеть содержимое всех полей выполняется переключение в режим формы.

Отображение данных в форме таблицы обеспечивает компонент DataGridView (рис. 5.19). Свойства компонента (табл. 5.8) определяют вид таблицы и действия, которые могут быть выполнены над данными во время работы программы.

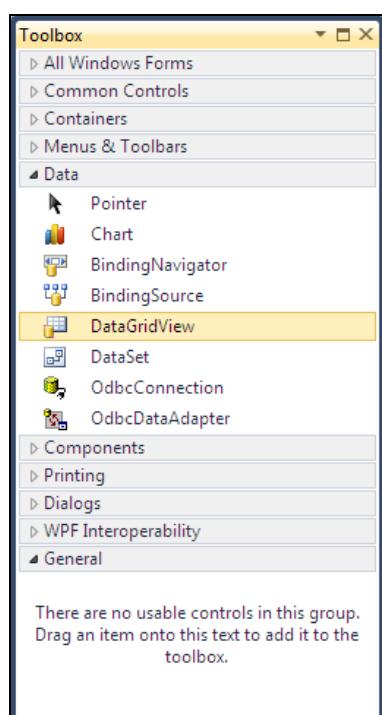


Рис. 5.19. Компонент DataGridView

Таблица 5.8. Свойства компонента DataGridView

Свойство	Описание
DataSource	Ссылка на источник данных (компонент DataSet)
DataMember	Ссылка на данные (таблицу из коллекции Tables компонента DataSet)
Columns	Отображаемая информация (столбцы) — коллекция элементов, каждый из которых задает вид столбца
ReadOnly	Запрещает (True) внесение изменений в таблицу
AllowsUsersToDeleteRows	Разрешает (True) удалять строки из таблицы
AllowsUsersToAddRows	Разрешает (True) добавлять строки в таблицу
ScrollBars	Отображаемые полосы прокрутки: None — не отображать; Vertical — только вертикальная; Horizontal — только горизонтальная; Both — обе

Таблица 5.8 (окончание)

Свойство	Описание
EditMode	Задает режим активизации процесса редактирования. Чтобы активизировать режим редактирования записи, надо нажать клавишу <F2> (режим EditOnF2) или <Enter> (режим EditOnEnter)
AllowUserToResizeColumns	Разрешает (True) менять во время работы программы ширину колонок таблицы
AllowUserToResizeRows	Разрешает (True) менять во время работы программы высоту строк таблицы

Свойство `Columns` компонента `DataGridView` представляет собой коллекцию объектов `DataGridViewColumn`, свойства которых (табл. 5.9) определяют информацию, отображаемую в колонке.

Таблица 5.9. Свойства объекта `DataGridViewColumn`

Свойство	Описание
<code>DataPropertyName</code>	Поле, содержимое которого отображается в столбце
<code>HeaderText</code>	Заголовок столбца
<code>Width</code>	Ширина столбца

Настройка компонента `DBGrid` выполняется следующим образом. Сначала в коллекцию `Columns` надо добавить столько элементов, сколько столбцов данных необходимо отобразить в поле компонента `DataGridView`. Для этого следует раскрыть окно редактора коллекции — щелкнуть на кнопке с тремя точками, которая находится в поле значения свойства `Columns`, или из контекстного меню компонента (оно появляется в результате щелчка правой кнопкой мыши в поле компонента) выбрать команду **Edit Columns**. В окне редактора коллекции надо сделать щелчок на кнопке **Add** и задать значения свойств добавленного элемента (рис. 5.20).

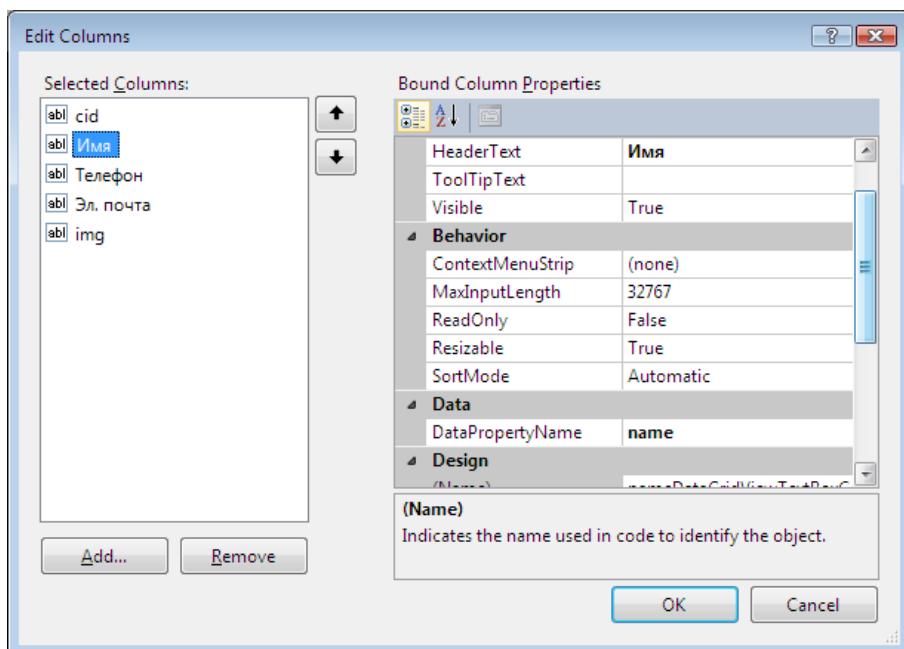
В табл. 5.10 приведены значения свойств компонента `DataGridView`, а на рис. 5.21 — вид формы после его настройки.

Таблица 5.10. Значения свойств компонента `DataGridView`

Свойство	Значение
<code>DataSource</code>	<code>dataSet1</code>
<code>DataMember</code>	<code>contacts</code>
<code>ScrollBars</code>	<code>Vertical</code>
<code>Size</code>	<code>435;155</code>
<code>AllowUserToResizeColumns</code>	<code>False</code>
<code>RowHeaderWidth</code>	<code>24</code>

Таблица 5.10 (окончание)

Свойство	Значение
Columns[0].HeaderText	cid
Columns[0].DataPropertyName	cid
Columns[0].Visible	False
Columns[1].HeaderText	Имя
Columns[1].DataPropertyName	name
Columns[1].Width	100
Columns[2].HeaderText	Телефон
Columns[2].DataPropertyName	phone
Columns[2].Width	100
Columns[3].HeaderText	Эл. почта
Columns[3].DataPropertyName	email
Columns[3].Width	100
Columns[4].HeaderText	img
Columns[4].DataPropertyName	img
Columns[4].Width	94

Рис. 5.20. Настройка элемента коллекции **Columns** компонента DataGridView

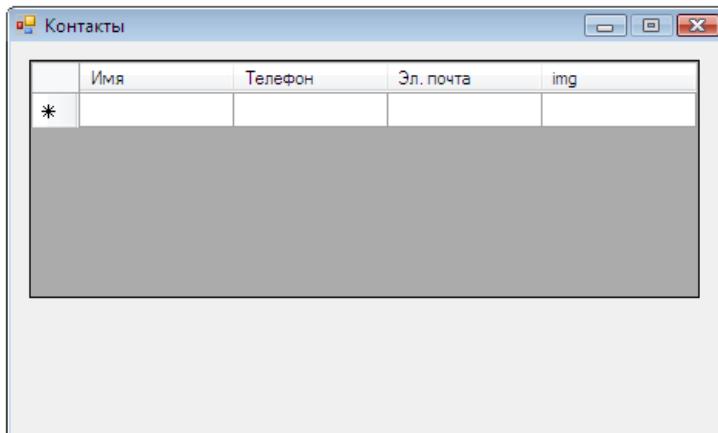


Рис. 5.21. Форма после настройки компонента DataGridView

Следующее, что надо сделать, — создать функции обработки событий FormLoad и FormClosing формы (листинг 5.1). Функция обработки события FormLoad должна загрузить данные, события FormClosing — сохранить изменения, сделанные пользователем. Загрузку данных выполняет метод Fill (в результате серверу направляется команда SELECT) компонента OleDbDataAdapter, которому в качестве параметра передается таблица объекта DataSet, заполняемая в результате выполнения команды. Следует обратить внимание на то, что все изменения, сделанные пользователем, автоматически фиксируются в базе данных в момент перехода к следующей записи. Однако если пользователь, не завершив, например, редактирование данных текущей записи, закроет окно программы, сделанные изменения не будут зафиксированы в базе данных. Поэтому, перед тем как завершить работу программы, надо принудительно обновить данные.

Листинг 5.1. База данных "Контакты"

```
// загрузка формы - начало работы программы
private void Form1_Load(object sender, EventArgs e)
{
    // прочитать данные из БД
    oleDbDataAdapter1.Fill(dataTable1);
}

// пользователь выщелкал строку и нажал <Delete>
private void dataGridView1_UserDeletingRow(object sender,
                                         DataGridViewRowCancelEventArgs e)
{
    DialogResult dr = MessageBox.Show(
        "Внимание! \nЗапись будет удалена из БД. \nВыполнить?",
```

```
"Удаление записи", MessageBoxButtons.OKCancel,
MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2);
if (dr == DialogResult.Cancel)
{
    e.Cancel = true;
}
}

// завершение работы программы
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    oleDbDataAdapter1.Update(dataSet1.Tables["contacts"]);
}
```

Выбор информации из базы данных

При работе с базой данных пользователя, как правило, интересует не все ее содержимое, а некоторая конкретная информация. В простейшем случае найти нужные сведения можно, просмотрев таблицу. Однако такой способ поиска неудобен и малоэффективен.

Выбрать нужную информацию из базы данных можно, направив серверу SQL-команду `SELECT`.

SQL-запрос

Чтобы выбрать из базы данных только нужные записи, надо направить серверу SQL-команду `SELECT`, указав в качестве параметра критерий отбора записей.

В общем виде SQL-команда `SELECT`, обеспечивающая выборку записей из базы данных (таблицы), выглядит так:

`SELECT СписокПолей FROM Таблица WHERE (Критерий) ORDER BY СписокПолей`

Параметр `Таблица` задает таблицу базы данных, из которой надо выбрать (получить) данные. Параметр `СписокПолей`, указанный после слова `SELECT`, задает поля, содержимое которых надо получить (если необходимы данные из всех полей, то вместо списка полей можно указать "звездочку"). Параметр `Критерий` задает критерий (условие) отбора записей. Параметр `СписокПолей`, указанный после `ORDER BY`, задает поля, по содержимому которых будут упорядочены записи таблицы, сформированной в результате выполнения команды.

Например, команда

`SELECT name, phone FROM contacts WHERE name = 'Никита Кульгин'`

обеспечивает выборку из таблицы `contacts` записи, у которой в поле `name` находится текст `Никита Кульгин`.

В критерии запроса (при сравнении строк) вместо конкретного значения можно указать шаблон. Например, шаблон `Ива%` обозначает все строки, которые начинаются с `Ива`, а шаблон `%Ива%` — все строки, в которых есть подстрока `Ива`. При исполь-

зовании шаблонов вместо оператора = надо использовать оператор LIKE. Например, запрос

```
SELECT * FROM contacts WHERE name LIKE 'Ку%'
```

выберет из таблицы contacts только те записи, в поле name которых находится текст, начинающийся с Ку. Вместо оператора LIKE можно использовать оператор CONTAINING (Содержит). Например, приведенный ранее запрос, целью которого является вывод списка людей, фамилии которых начинаются с Ку, при использовании оператора CONTAINING будет выглядеть так:

```
SELECT * FROM contacts WHERE name CONTAINING 'Ку'
```

Если запрос формируется во время работы программы, то следует использовать параметры. Параметр в SQL-команде обозначается символом "?". Например, команда с параметром, обеспечивающая выборку из таблицы contacts записей по содержимому поля name, выглядит так:

```
SELECT * FROM contacts WHERE name LIKE ?
```

Очевидно, что перед тем как активизировать выполнение команды, параметру надо присвоить значение.

Здесь надо вспомнить, что доступ к базе данных обеспечивает компонент OleDbDataAdapter, свойство SelectCommand которого представляет собой объект OleDbCommand, а свойство Parameters объекта OleDbCommand является списком параметров команды. Следует обратить внимание на то, что независимо от числа параметров команды все они обозначаются символом "?". Связь между параметрами в команде и в списке параметров осуществляется по номеру. Первому параметру соответствует нулевой элемент коллекции Parameters (элементы коллекции numеруются с нуля), второму — первый и т. д.

Процесс выборки информации из базы данных с использованием SQL-запроса с параметром демонстрирует программа "Контакты-2" (ее форма приведена на рис. 5.22).

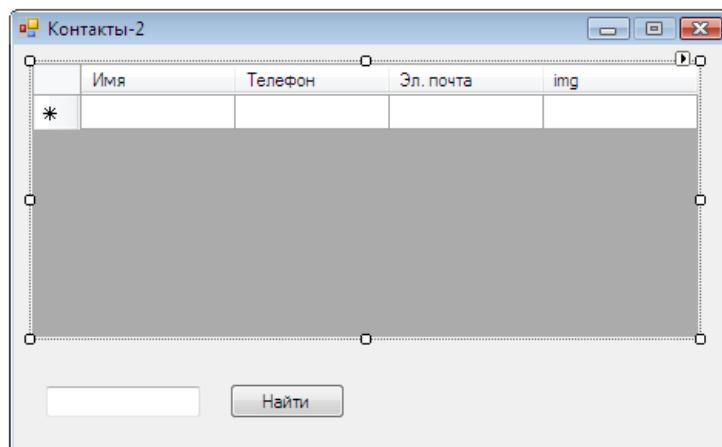


Рис. 5.22. Форма программы "Контакты-2"

Программа "Контакты-2" создается путем модернизации программы "Контакты": на форму надо добавить поле редактирования и командную кнопку. Кроме того, надо изменить настройку компонента OleDbDataAdapter (табл. 5.11). Функции обработки событий приведены в листинге 5.2. Выбор информации из базы данных выполняет функция обработки события Click для кнопки **Найти**. Она формирует SQL-запрос и направляет его серверу.

Таблица 5.11. Значения свойств компонента OleDbDataAdapter

Свойство	Значение
SelectCommand.Connection	oleDbConnection1
SelectCommand.CommandText	SELECT * FROM contacts WHERE name LIKE ?
SelectCommand.Parameters[0].ParameterName	name
SelectCommand.Parameters[0].SourceColumn	name
SelectCommand.Parameters[0].Value	%%

Листинг 5.2. Выбор информации из БД

```
// начало работы программы
private void Form1_Load(object sender, EventArgs e)
{
    // Получить информацию из БД.
    // У команды SELECT есть параметр, который задает
    // критерий отбора записей, поэтому надо задать его значение
    oleDbTypeAdapter1.SelectCommand.Parameters[0].Value = "%%";
    oleDbTypeAdapter1.Fill(dataTable1);
}

// щелчок на кнопке Найти
private void button1_Click(object sender, EventArgs e)
{
    dataSet1.Clear(); // удалить старые данные

    // Для получения информации из базы данных
    // используется команда SELECT с параметром:
    // SELECT *FROM contacts WHERE (name Like ?)
    // где: ? - параметр
    // В программе доступ к параметру можно получить по номеру или по имени.

    // Задать значение параметра команды SELECT
    // Доступ по номеру
    // oleDbTypeAdapter1.SelectCommand.Parameters[0].Value =
    //           "%" + textBox1.Text + "%";
}
```

```
// доступ к параметру по имени
oleDbTypeAdapter1.SelectCommand.Parameters["name"].Value =
    "%" + textBox1.Text + "%";

// выполнить команду
oleDbTypeAdapter1.Fill(dataTable1);

}

// пользователь нажал клавишу <Delete>
private void dataGridView1_UserDeletingRow(object sender,
                                         DataGridViewRowCancelEventArgs e)
{
    DialogResult r;

    r = MessageBox.Show(
        "Вы действительно хотите удалить запись?", "Удаление записи",
        MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation,
        MessageBoxDefaultButton.Button2);

    if (r == DialogResult.Cancel)
    {
        // отменить операцию удаления записи
        e.Cancel = true;
    }
}

// завершение работы программы
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    try
    {
        oleDbTypeAdapter1.Update(dataTable1);
    }
    catch (Exception e1)
    {
        MessageBox.Show(e1.ToString());
    }
}
```

Работа с базой данных в режиме формы

На практике используются два режима отображения данных: таблица и форма.

В режиме таблицы в окне программы отображается таблица, что позволяет видеть несколько записей одновременно. Обычно этот режим применяется для просмотра записей. Отображение данных в режиме таблицы обеспечивает компонент DBGrid. Если в таблице, содержимое которой отображается в поле компонента

DBGrid, много колонок, то пользователь, как правило, не может видеть все столбцы одновременно, и, для того чтобы увидеть нужную информацию, он вынужден менять ширину столбцов или прокручивать содержимое поля компонента по горизонтали, что не совсем удобно.

В режиме формы в окне программы отображается только одна запись, что позволяет одновременно видеть содержимое *всех* полей записи. Обычно режим формы используется для ввода информации в базу данных, а также для просмотра записей, состоящих из большого количества полей. Часто режим формы и режим таблицы комбинируют.

Для отображения (редактирования) полей записей базы данных можно использовать компонент TextBox. Чтобы в поле компонента TextBox отображалось содержимое поля записи базы данных, его надо *связать* с соответствующим элементом набора данных — столбцом таблицы компонента DataSet. Связывание осуществляется путем установки значений свойств DataBinding.Text компонента TextBox. Связать компонент с источником данных можно во время создания формы (рис. 5.23) или во время работы программы, добавив в конструктор соответствующие инструкции.

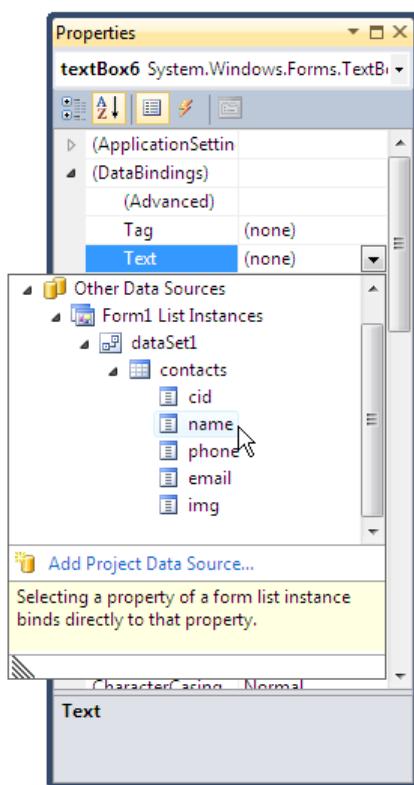


Рис. 5.23. Связывание компонента TextBox с источником данных

В качестве примера рассмотрим программу, которая обеспечивает работу с базой "Контакты" в режиме формы.

Форма программы работы с БД "Контакты" приведена на рис. 5.24. В полях textBox1—textBox3 отображается соответственно содержимое полей name, phone и email текущей записи. Следует обратить внимание на то, что за компонентом pictureBox1 находится "спрятан" компонент textBox4. В нем находится имя файла иллюстрации (содержимое поля img) отображаемой в поле pictureBox1.

Функции обработки событий приведены в листинге 5.3.

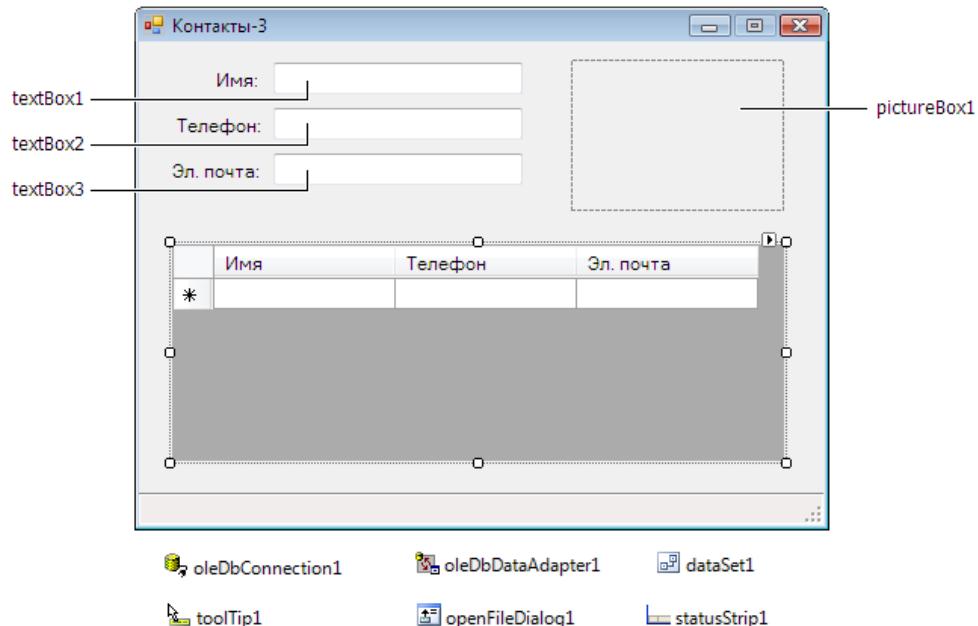


Рис. 5.24. Форма программы работы с базой данных "Контакты" (режим формы)

Листинг 5.3. База данных "Контакты" (режим формы)

```

public Form1()
{
    InitializeComponent();
}

string dbPath; // папка базы данных
string imFolder; // папка иллюстраций

// начало работы программы
private void Form1_Load(object sender, EventArgs e)
{
    // загрузить путь к файлу БД из файла конфигурации
    //dbPath = Settings.Default.DbPath;

    // файл базы данных находится в папке Документы\Contacts
    dbPath = Environment.GetFolderPath
        (Environment.SpecialFolder.MyDocuments) + "\\Contacts";

    // открыть соединение с БД
    oleDbConnection1.ConnectionString =
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
        dbPath + "\\Contacts.mdb";
}

```

```
// папка иллюстраций
imFolder = dbPath + "\\Images\\";

try
{
    // получить информацию из БД
    oleDbDataAdapter1.Fill(dataSet1.Tables[0]);

    // показать имя файла БД в строке состояния
    toolStripStatusLabel1.Text = dbPath + "\\Contacts\\contacts.mdb";
}

catch (Exception e1)
{
    MessageBox.Show("Не найден файл БД\n" + dbPath, "Контакты_3",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
    textBox1.Enabled = false;
    textBox2.Enabled = false;
    textBox3.Enabled = false;
    dataGridView1.Enabled = false;
    toolStripStatusLabel1.Text = "";
    pictureBox1.Enabled = false;
}

// пользователь нажал клавишу <Delete>
private void dataGridView1_UserDeletingRow(object sender,
                                         DataGridViewRowCancelEventArgs e)
{
    DialogResult r;

    r = MessageBox.Show("Вы действительно хотите удалить запись?",
                       "Удаление записи",
                       MessageBoxButtons.OKCancel,
                       MessageBoxIcon.Exclamation,
                       MessageBoxDefaultButton.Button2);

    if (r == DialogResult.Cancel)
    {
        // отменить операцию удаления записи
        e.Cancel = true;
    }
}

// Изменилось содержимое поля textBox4 -
// отобразить иллюстрацию, имя файла которой находится в этом поле
private void textBox4_TextChanged(object sender, EventArgs e)
```

```
{  
    string imageFile;  
    string msg; // текст, отображаемый в поле компонента pictureBox  
  
    if (textBox4.Text.Length == 0)  
    {  
        imageFile = imFolder + "nobody.jpg";  
    }  
    else  
        imageFile = imFolder + textBox4.Text;  
  
    try  
    {  
        msg = "";  
        pictureBox1.Image = System.Drawing.Bitmap.FromFile(imageFile);  
    }  
    catch (System.IO.FileNotFoundException)  
    {  
        // вывести сообщение об ошибке в поле компонента pictureBox1  
        msg = "File not found: " + imageFile;  
        pictureBox1.Image = null;  
        pictureBox1.Refresh();  
    }  
}  
  
// щелчок в поле отображения иллюстрации  
private void pictureBox1_Click(object sender, EventArgs e)  
{  
    openFileDialog1.Title = "Выберите иллюстрацию";  
    openFileDialog1.InitialDirectory = imFolder;  
    openFileDialog1.Filter = "фото (*.jpg)|все файлы (*.*)";  
    openFileDialog1.FileName = "";  
  
    if (openFileDialog1.ShowDialog() == DialogResult.OK)  
    {  
        // пользователь указал файл иллюстрации  
  
        // проверим, находится ли выбранный файл в каталоге imFolder  
        bool r = openFileDialog1.FileName.ToLower().Contains(  
            openFileDialog1.InitialDirectory.ToLower());  
        if (r == true)  
        {  
            // копировать не надо, т. к. пользователь указал иллюстрацию,  
            // которая находится в imFolder  
            textBox4.Text = openFileDialog1.SafeFileName;  
        }  
    }  
}
```

```
else
{
    // скопировать файл иллюстрации в папку Images

    // если в каталоге-приемнике есть файл с таким же
    // именем, что и копируемый, возникает исключение
    try
    {
        // копировать файл
        System.IO.File.Copy(openFileDialog1.FileName,
                            imFolder + openFileDialog1.SafeFileName);

        textBox4.Text = openFileDialog1.SafeFileName;
    }
    catch (Exception ex)
    {
        DialogResult dr;
        dr = MessageBox.Show(ex.Message + " Заменить его?", "",
                            MessageBoxButtons.OKCancel,
                            MessageBoxIcon.Warning,
                            MessageBoxDefaultButton.Button2);
        if (dr == DialogResult.OK)
        {
            // перезаписать файл - overwrite = true
            System.IO.File.Copy(openFileDialog1.FileName,
                                imFolder + openFileDialog1.SafeFileName, true);
            textBox4.Text = openFileDialog1.SafeFileName;
        }
    }
}

}

// завершение работы программы
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    try
    {
        oleDbDataAdapter1.Update(dataTable1);
    }
    catch (Exception e1)
    {
        MessageBox.Show(e1.ToString());
    }
}
```

Отображение иллюстрации выполняет функция обработки события `TextChanged` компонента `textBox4`. Если в `textBox4` есть данные (поле `img` текущей записи не пустое), то в `pictureBox1` отображается соответствующая картинка, а если поле `img` текущей записи пустое, то отображается картинка `nobody.jpg`.

Информация в поля `name`, `phone` и `email` текущей записи вводится обычным образом — путем заполнения полей **Имя**, **Телефон** и **Эл. почта** или вводом данных в таблицу. Чтобы ввести информацию в поле `img` (задать имя файла иллюстрации), надо сделать щелчок в поле компонента `PictureBox` и в открывшемся окне **Выбор изображения** (отображение диалога обеспечивает компонент `OpenFileDialog`) указать файл иллюстрации. Следует обратить внимание на то, что имя файла иллюстрации записывается в поле `img`, а сам файл копируется в каталог `Images`.

Сервер баз данных Microsoft SQL Server Compact Edition

Сервер Microsoft SQL Server Compact Edition представляет собой компактный, высокопроизводительный сервер баз данных. Но он, в отличие от Microsoft SQL Server, не требует администрирования, что делает его использование в информационных системах среднего уровня наилучшим решением.

На компьютер разработчика Microsoft SQL Server Compact Edition устанавливается обычным образом — путем запуска установщика (установочный файл Microsoft SQL Server Compact Edition можно загрузить с сайта Microsoft).

Среда Microsoft SQL Server Management Studio

Для работы с сервером Microsoft SQL Server Compact Edition удобно использовать среду Microsoft SQL Server Management Studio (версию Express можно бесплатно загрузить с сайта Microsoft). С ее помощью можно создать базу данных, наполнить ее информацией, направить серверу запрос и увидеть результат его выполнения.

На компьютер Microsoft SQL Server Management Studio устанавливается обычным образом — путем запуска установщика (установочный файл Microsoft SQL Server Management Studio можно загрузить с сайта Microsoft).

Создание базы данных

Создать базу данных Microsoft SQL Server Compact Edition можно при помощи Microsoft SQL Server Management Studio или программно, направив серверу соответствующий запрос (см. листинг 5.5).

Процесс создания базы Microsoft SQL Server Compact Edition в Microsoft SQL Server Management Studio рассмотрим на примере. Создадим базу данных "Контакты".

Сначала надо запустить Microsoft SQL Server Management Studio и в раскрывающемся списке **Тип сервера** выбрать **SQL Server Compact Edition** (рис. 5.25).

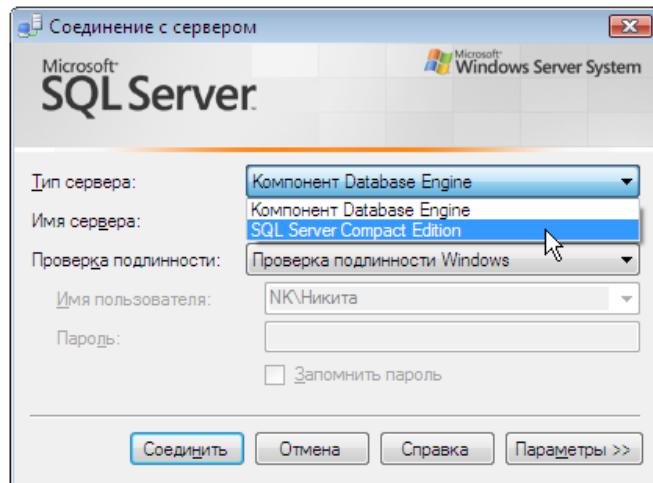


Рис. 5.25. Выбор сервера, к которому надо подключиться

Далее в списке **Файл базы данных** надо выбрать **Новая база данных**, затем (рис. 5.26) в окне **Create New SQL Server Compact Edition Database** ввести имя файла базы данных (рис. 5.27) и, если необходимо ограничить доступ к базе данных, пароль. После нажатия кнопки **OK** во вновь ставшем доступным окне **Соединение с сервером** (рис. 5.28) надо нажать кнопку **Соединить**.

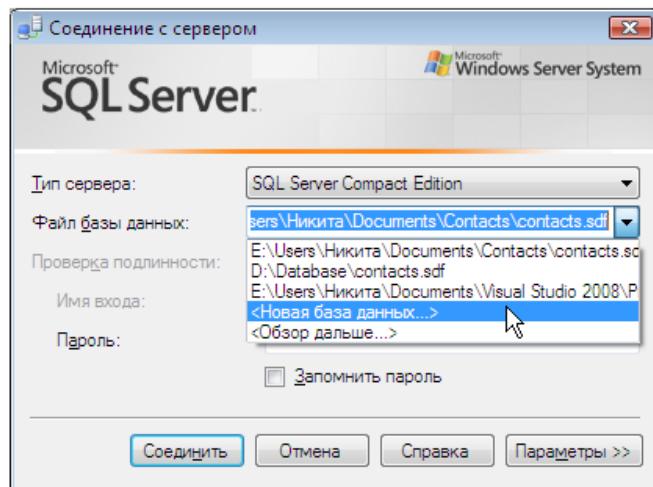


Рис. 5.26. Создание файла базы данных Microsoft SQL Server Compact Edition (шаг 1)

В результате описанных действий будет создан файл базы данных и установлено соединение с созданной базой данных. Окно среды SQL Server Management Studio показано на рис. 5.29.

Чтобы создать в базе данных таблицу, надо в окне **Обозреватель объектов**, в контекстном меню раздела **Таблицы**, выбрать команду **Создать таблицу**, а затем в появившемся окне задать имя таблицы (поле **Name**) и определить ее структуру

(рис. 5.29). В поле **Column Name** надо ввести имя столбца, в поле **Data Type** — тип данных, в поле **Length** для полей строкового типа (`nvarchar` — строка символов переменной длины) надо ввести максимально допустимую длину строки. Если по условию решаемой задачи поле обязательно должно содержать информацию, то в поле **Allow Nulls** надо ввести `No`.

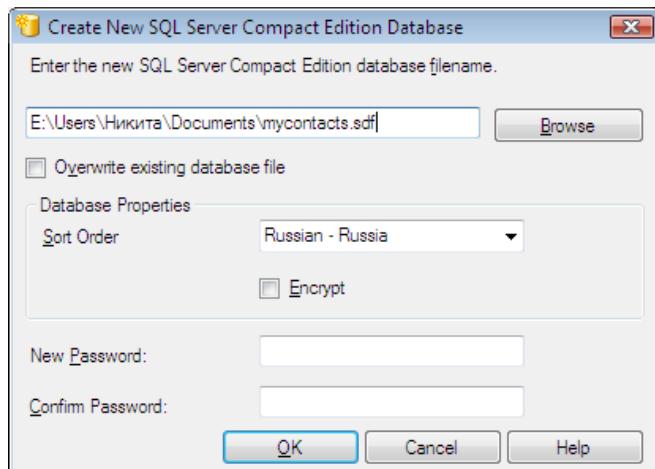


Рис. 5.27. Создание файла базы данных Microsoft SQL Server Compact Edition (шаг 2)

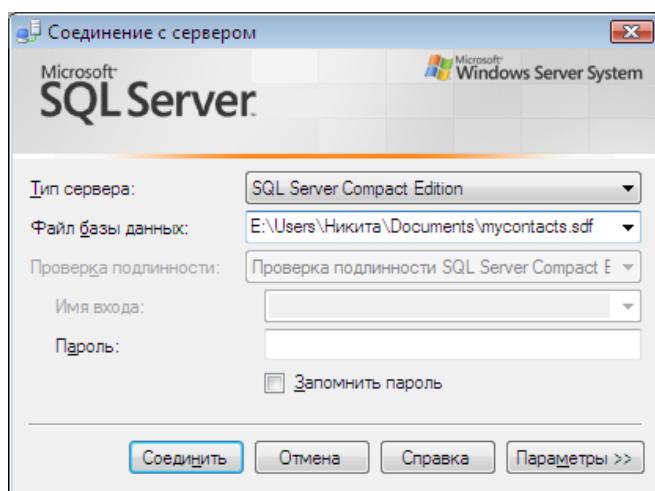


Рис. 5.28. Создание файла базы данных Microsoft SQL Server Compact Edition (шаг 3)

Обратите внимание на свойство `Identity` столбца `cid`. Значение `True` показывает, что поле `cid` используется в качестве уникального идентификатора записи. Идентификатор автоматически формируется при добавлении в таблицу записи путем увеличения на единицу (`IdentityIncrement = 1`) идентификатора последней добавленной записи. Значение `IdentitySeed` задает идентификатор первой записи таблицы.

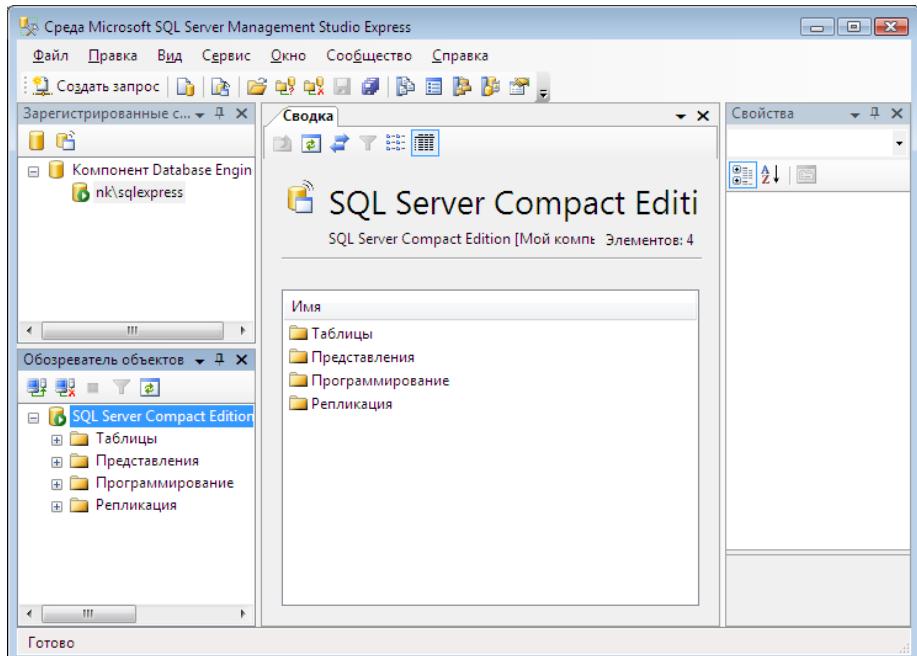


Рис. 5.29. Среда SQL Server Management Studio обеспечивает управление сервером

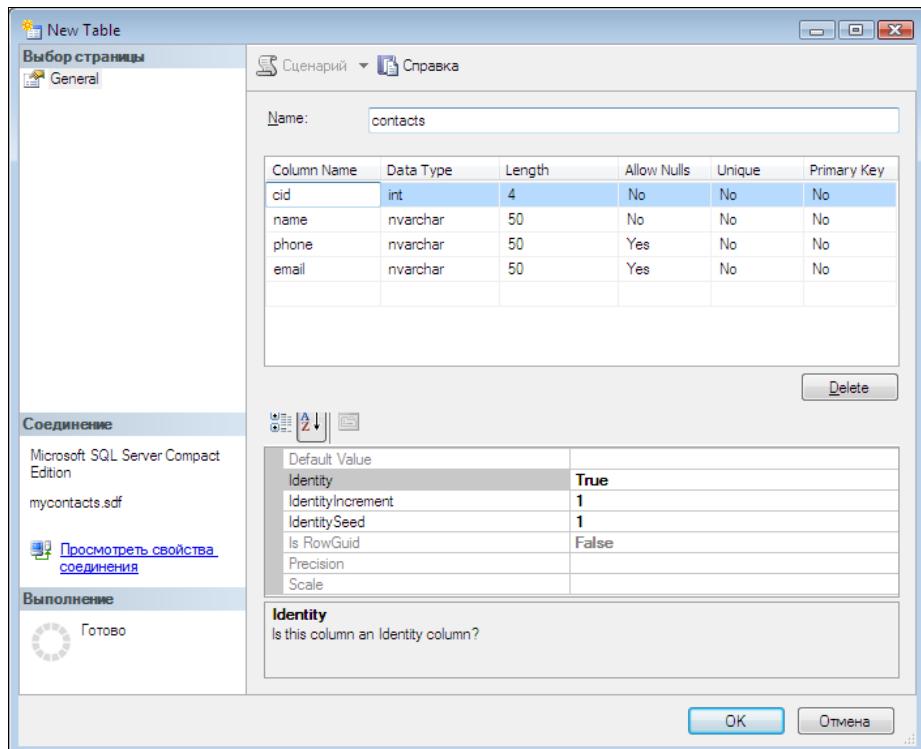


Рис. 5.30. Создание таблицы в базе данных

Таблицу в базе данных можно создать так же, направив серверу соответствующий запрос (SQL-команду). Для этого надо в меню **Файл** выбрать команду **Создать > Запрос в текущем соединении** и в открывшемся окне набрать SQL-команду, обеспечивающую создание таблицы (рис. 5.31). Чтобы направить запрос серверу, надо в меню **Запрос** выбрать команду **Выполнить**.

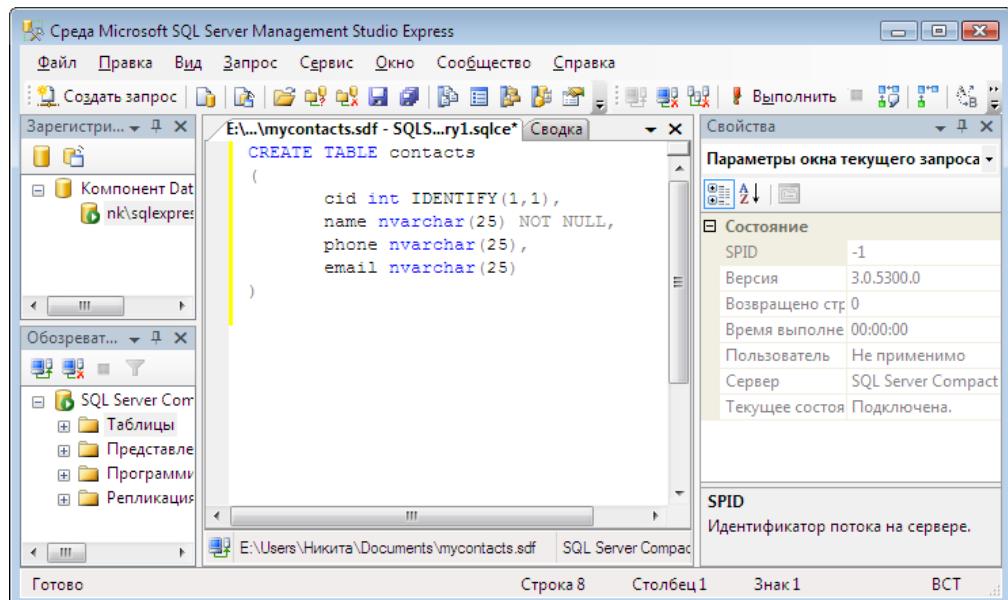


Рис. 5.31. Команда создания таблицы

После того как база данных будет создана, можно приступить к созданию приложения.

База данных "Контакты"

Процесс создания приложения работы с базой данных Microsoft SQL Server Compact Edition рассмотрим на примере — создадим программу работы с базой данных "Контакты".

Перед тем как приступить к работе над программой, надо при помощи SQL Server Management Studio создать базу данных (файл mycontacts.mdf) и поместить в нее таблицу contacts (табл. 5.12). SQL-команды, обеспечивающие создание таблицы, приведены в листинге 5.4.

Таблица 5.12. Поля таблицы contacts

Поле	Тип	Примечание
cid	INT	Автоувеличение
name	nvarchar(50)	Обязательное поле (Not null)

Таблица 5.12 (окончание)

Поле	Тип	Примечание
phone	nvarchar(50)	
email	nvarchar(50)	

Листинг 5.4. Создание таблицы contacts

```
CREATE TABLE contacts
(
    cid int IDENTITY(1,1),
    name nvarchar(50) NOT NULL,
    phone nvarchar(50),
    email nvarchar(50)
)
```

Форма программы работы с базой данных Microsoft SQL Server Compact Edition "Контакты" приведена на рис. 5.32. Для отображения записей в табличной форме используется компонент ListView (значения его свойств приведены в табл. 5.13). Компоненты textBox1, textBox2 и textBox3 используются для ввода и редактирования полей name, phone и email, а компонент textBox4 (свойству ReadOnly которого следует присвоить значение True) — для хранения значения поля cid. Нетрудно заметить, что на форме нет компонентов доступа к базе данных. Все необходимые для работы с базой данных объекты создаются "в коде", во время работы программы.

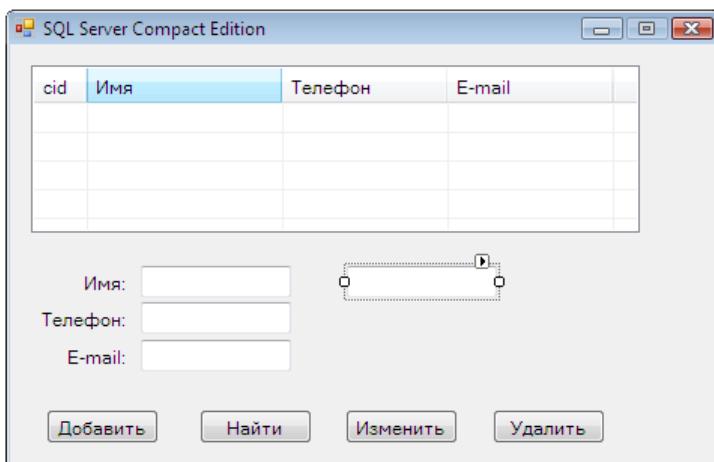


Рис. 5.32. Форма программы работы с базой данных Microsoft SQL Server Compact Edition

Таблица 5.13. Значения свойств компонента ListView

Свойство	Значение
Columns[0].Text	cid
Columns[0].Width	35
Columns[1].Text	Имя
Columns[1].Width	130
Columns[2].Text	Телефон
Columns[2].Width	110
Columns[3].Text	E-mail
Columns[3].Width	110
View	Details
HideSelection	False
MultiSelect	False
GridLines	True
FullRowSelect	True

Доступ к базе данных Microsoft SQL Server Compact Edition обеспечивают объекты SqlCeEngine, SqlCeConnection, SqlCeDataAdapter, SqlCeCommand, SqlCeDataReader, принадлежащие пространству имен System.Data.SqlServerCe. Чтобы это пространство имен стало доступно, в проект надо добавить ссылку на сборку System.Data.SqlClient.dll, в которой оно определено. Для этого надо в меню **Project** выбрать команду **Add Reference**, раскрыть вкладку .NET и указать сборку (рис. 5.33).

При переносе программы на другой компьютер по умолчанию предполагается, что все необходимые для ее работы сборки (библиотеки) на этом компьютере есть. Как правило, это так. Однако если программа использует "нестандартные" библиотеки, сборки, которые по умолчанию в .NET Framework (точнее, в Global Assembly Cache, GAC) не входят, то надо позаботиться об установке на компьютер пользователя необходимых программе сборок. Чтобы подключенная сборка гарантированно была доступна программе при переносе ее на другой компьютер, свойству **Copy Local** надо присвоить значение **True** (рис. 5.34). В этом случае при построении программы она будет скопирована в каталог выполняемой программы (в зависимости от текущего режима компиляции в папку Debug или Release).

Если нужная сборка находится не в GAC (имя сборки на вкладке .NET не отображается), то необходимо раскрыть вкладку **Browse** и указать файл сборки там (в этом случае файл сборки во время компиляции будет скопирован в каталог проекта).

Конструктор формы и функции обработки событий приведены в листинге 5.5.

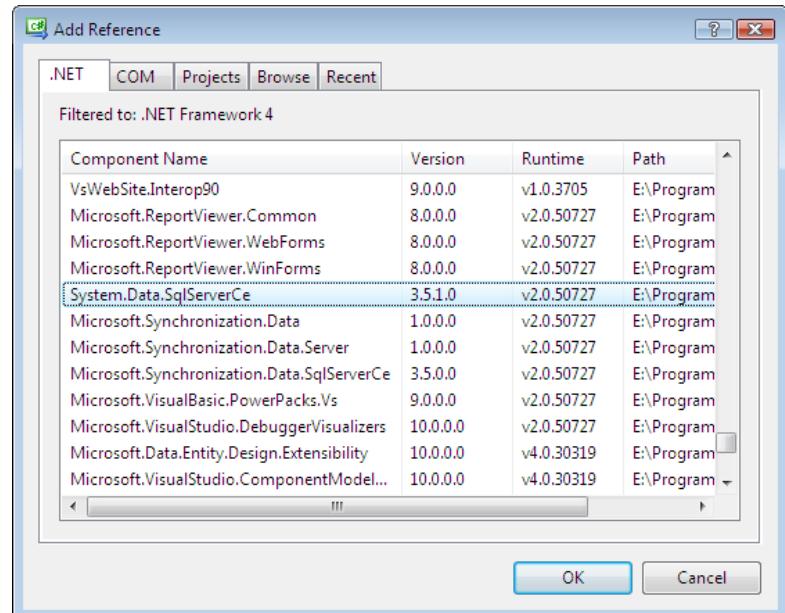
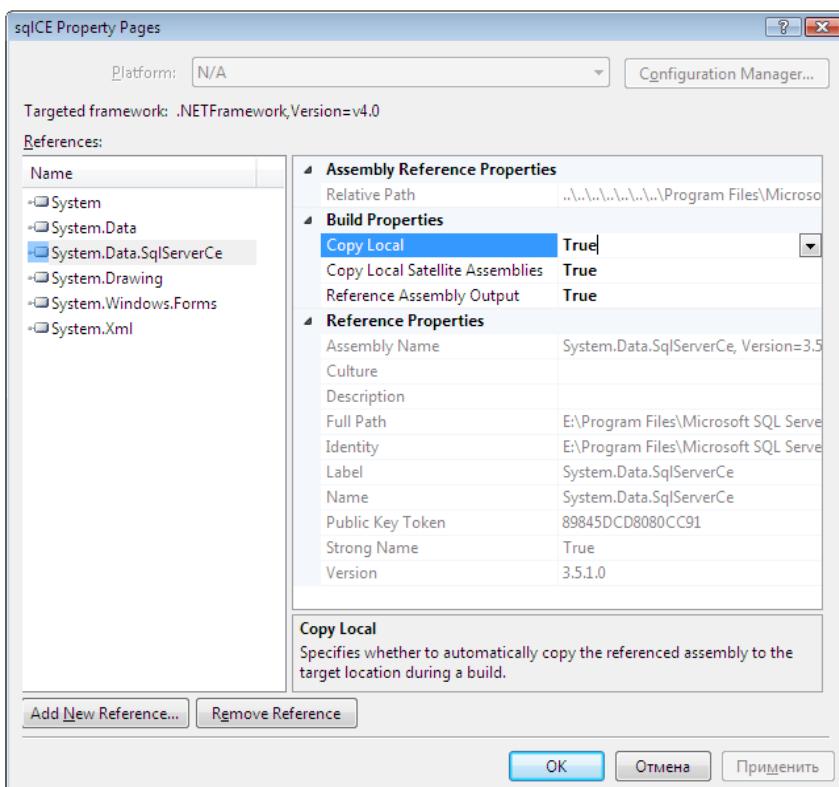


Рис. 5.33. Добавление в проект ссылки на сборку

Рис. 5.34. Чтобы сборка гарантированно была доступна программе при переносе ее на другой компьютер, свойству **Copy Local** надо присвоить значение **True**

Листинг 5.5. База данных Microsoft SQL Server CE "Контакты"

```
/*
Чтобы пространство имен System.Data.SqlClient стало доступно,
в проект надо добавить ссылку на файл сборки, в котором оно
определенено. Для этого: в меню Project выберите команду Add Reference
и на вкладке .NET укажите сборку System.Data.SqlClient.dll
*/
InitializeComponent();

// настройка компонента ListView:
// увеличим ширину компонента на 17 - ширину полосы прокрутки
int w = 0;
for (int i = 0; i < listView1.Columns.Count; i++)
{
    w += listView1.Columns[i].Width;
}

if (listView1.BorderStyle == BorderStyle.FixedSingle)
    w += 4;

listView1.Width = w+17; // 17 - область отображения полосы прокрутки

// выделять всю строку (элемент и подэлементы)
listView1.FullRowSelect = true;
}

private void Form1_Load(object sender, EventArgs e)
{
    // Создать БД SQL Server Compact Edition.
    // Если путь к файлу не указан, БД будет создана в каталоге приложения
    SqlCeEngine engine;

    engine = new SqlCeEngine("Data Source='contacts.sdf';");
    if (!(File.Exists("contacts.sdf")))
    {
        engine.CreateDatabase();
        SqlCeConnection connection =
            new SqlCeConnection(engine.LocalConnectionString);
        connection.Open();
        SqlCeCommand command = connection.CreateCommand();
        command.CommandText =
            "CREATE TABLE contacts (cid int IDENTITY(1,1), name nvarchar(50)
NOT NULL, phone nvarchar(50), email nvarchar(50))";
        command.ExecuteScalar();
        connection.Close();
    }
}
```

```
else
{
    ShowDB();
}
}

private void ShowDB()
{
    SqlCeEngine engine = new SqlCeEngine("Data Source='contacts.sdf';");
    SqlCeConnection connection =
        new SqlCeConnection(engine.LocalConnectionString);
    connection.Open();
    SqlCeCommand command = connection.CreateCommand();
    command.CommandText = "SELECT * FROM contacts ORDER BY name";
    SqlCeDataReader dataReader = command.ExecuteReader();

    string st; // значение поля БД
    int itemIndex = 0;

    listView1.Items.Clear();

    while (dataReader.Read())
    {
        for (int i = 0; i < dataReader.FieldCount; i++)
        {
            st = dataReader.GetValue(i).ToString();
            switch (i)
            {
                case 0: // поле cid
                    listView1.Items.Add(st);
                    break;
                case 1: // поле name
                    listView1.Items[itemIndex].SubItems.Add(st);
                    //listView1.Items.Add(st);
                    break;
                case 2: // поле phone
                    listView1.Items[itemIndex].SubItems.Add(st);
                    break;
                case 3: // поле email
                    listView1.Items[itemIndex].SubItems.Add(st);
                    break;
            };
        }
        itemIndex++;
    }
    connection.Close();
}
```

```
// пользователь выбрал строку в поле компонента listView
private void listView1_SelectedIndexChanged(object sender,
ListViewItemSelectionChangedEventArgs e)
{
    // При выборе строки событие ItemSelectionChanged возникает два раза:
    // первый раз, когда выделенная в данный момент строка теряет фокус,
    // второй - когда строка, в которой сделан щелчок, получает фокус.
    // Нас интересует строка, которая получает фокус.

    if (e.IsSelected)
    {
        // строка выбрана, т. е. она получила фокус
        textBox4.Text = listView1.Items[e.ItemIndex].Text;

        for (int i = 1; i < listView1.Items[e.ItemIndex].SubItems.Count; i++)
        {
            switch (i)
            {
                case 1:
                    textBox1.Text =
                        listView1.Items[e.ItemIndex].SubItems[i].Text;
                    break;
                case 2:
                    textBox2.Text =
                        listView1.Items[e.ItemIndex].SubItems[i].Text;
                    break;
                case 3:
                    textBox3.Text =
                        listView1.Items[e.ItemIndex].SubItems[i].Text;
                    break;
            }
        }
    }

    // щелчок на кнопке Добавить
    private void button1_Click(object sender, EventArgs e)
    {
        SqlCeConnection conn =
            new SqlCeConnection("Data Source ='contacts.sdf'");
        conn.Open();
        SqlCeCommand command = conn.CreateCommand();
        command.CommandText =
            "INSERT INTO contacts(name, phone,email) VALUES (?,?,?)";
        command.Parameters.Add("name", textBox1.Text);
        command.Parameters.Add("phone", textBox2.Text);
        command.Parameters.Add("email", textBox3.Text);
        command.ExecuteScalar();
        conn.Close();
    }
}
```

```

// очистить поля ввода
textBox1.Clear();
textBox2.Clear();
textBox3.Clear();

ShowDB();

// установить курсор в поле textBox1
textBox1.Focus();
}

// щелчок на кнопке Найти
private void button2_Click(object sender, EventArgs e)
{
    SqlCeEngine engine = new SqlCeEngine("Data Source='contacts.sdf';");
    SqlCeConnection connection =
        new SqlCeConnection(engine.LocalConnectionString);
    connection.Open();

    SqlCeCommand command = connection.CreateCommand();

    command.CommandText = "SELECT * FROM contacts WHERE (name LIKE ?)";
    command.Parameters.Add("name", "%" + textBox1.Text + "%");

    SqlCeDataReader dataReader = command.ExecuteReader();

    string st; // значение поля БД
    int itemIndex = 0;

    listView1.Items.Clear();

    while (dataReader.Read())
    {
        for (int i = 0; i < dataReader.FieldCount; i++)
        {
            st = dataReader.GetValue(i).ToString();
            switch (i)
            {
                case 0: // поле cid
                    listView1.Items.Add(st);
                    break;
                case 1: // поле name
                    //listView1.Items.Add(st);
                    listView1.Items[itemIndex].SubItems.Add(st);
                    break;
                case 2: // поле phone
                    listView1.Items[itemIndex].SubItems.Add(st);
                    break;
            }
        }
    }
}

```

```
        case 3: // поле email
            listView1.Items[itemIndex].SubItems.Add(st);
            break;
        };
    }
    itemIndex++;
}
connection.Close();
}

// щелчок на кнопке Удалить
private void button3_Click(object sender, EventArgs e)
{
    if (listView1.SelectedItems.Count != 0)
    {
        //MessageBox.Show(listView1.SelectedItems[0].Text);

        SqlCeEngine engine =
            new SqlCeEngine("Data Source='contacts.sdf';");
        SqlCeConnection connection =
            new SqlCeConnection(engine.LocalConnectionString);
        connection.Open();

        SqlCeCommand command = connection.CreateCommand();

        command.CommandText = "DELETE FROM contacts WHERE (cid = ?)";
        command.Parameters.Add("cid", textBox4.Text);

        command.ExecuteScalar(); // выполнить команду

        ShowDB();

        textBox1.Clear();
        textBox2.Clear();
        textBox3.Clear();
        textBox4.Clear();
    }
}

// щелчок на кнопке Изменить
private void button4_Click(object sender, EventArgs e)
{
    if (listView1.SelectedItems.Count != 0)
    {
        SqlCeEngine engine =
            new SqlCeEngine("Data Source='contacts.sdf';");
        SqlCeConnection connection =
```

```

new SqlCeConnection(engine.LocalConnectionString);
connection.Open();

SqlCeCommand command = connection.CreateCommand();

command.CommandText =
    "UPDATE contacts SET name = ?, phone =?, email=? WHERE cid = ?";
command.Parameters.Add("name", textBox1.Text);
command.Parameters.Add("phone", textBox2.Text);
command.Parameters.Add("email", textBox3.Text);
command.Parameters.Add("cid", textBox4.Text);

command.ExecuteScalar(); // выполнить команду

ShowDB();

textBox1.Clear();
textBox2.Clear();
textBox3.Clear();
textBox4.Clear();

}

}

// изменился текст в поле редактирования
private void textBox1_TextChanged(object sender, EventArgs e)
{
    // кнопка Добавить становится доступной, если информация введена
    // в поле Имя и в какое-либо из полей Телефон или E-mail
    if ((textBox1.TextLength > 0) &&
        ((textBox2.TextLength > 0) || (textBox3.TextLength > 0)))
        button1.Enabled = true;

    else
    {
        button1.Enabled = false;
    }
}

```

Развертывание приложения работы с базой данных Microsoft SQL Server Compact Edition

Для того чтобы программа работы с базой данных Microsoft SQL Server Compact Edition могла работать на другом компьютере, на нем, помимо .NET Framework соответствующей версии, должен быть установлен сервер баз данных Microsoft SQL Server Compact Edition, который физически представляет собой .NET-сборку System.Data.SqlServerCe.dll. Поместить файл сборки надо в тот каталог, в котором находится приложение работы с базой данных.

ГЛАВА 6

Консольное приложение

надо Консольное приложение — это программа, которая для взаимодействия с пользователем не использует графический интерфейс. Для взаимодействия с пользователем используется *консоль* — клавиатура и монитор, работающий в режиме отображения символьной информации (буквы, цифры и специальные знаки).

В операционной системе консольное приложение работает в окне командной строки (рис. 6.1), которое часто называют окном консоли.

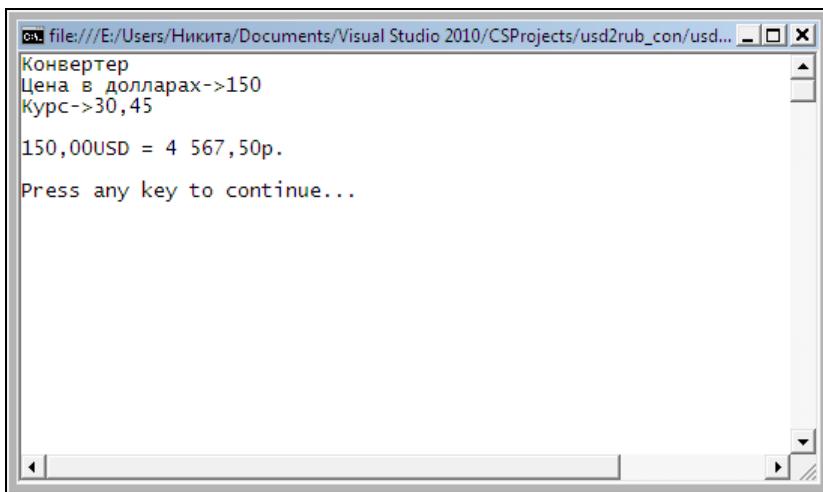


Рис. 6.1. Консольное приложение работает в окне командной строки

Консольные приложения удобны для решения задач, в которых не предъявляются особые требования к интерфейсу. Они широко используются для решения системных задач. Следует обратить внимание, что многие утилиты Microsoft .NET Framework реализованы как консольные приложения.

Консольное приложение может вывести информацию на экран и получить данные с клавиатуры одним из трех способов:

- ◆ при помощи функций `printf` (вывод) и `scanf` (ввод);
- ◆ вывести информацию в поток вывода (`cout`), прочитать данные из потока ввода (`cin`);
- ◆ при помощи методов `WriteLine` и `ReadLine` объекта `Console`.

Основным способом взаимодействия с пользователем в консольных .NET-приложениях, созданных Microsoft Visual Studio, является использование объекта `Console`. Методы, обеспечивающие отображение и ввод данных, перечислены в табл. 6.1.

Таблица 6.1. Методы объекта `Console`

Метод	Описание
<code>Console.WriteLine(st)</code>	Выводит на экран (в окно консоли) строку <code>st</code>
<code>Console.ReadLine(st)</code>	Выводит на экран (в окно консоли) строку <code>st</code> , после чего переводит курсор в начало следующей строки
<code>Console.ReadLine()</code>	Значением метода <code>ReadLine</code> является строка, набранная пользователем на клавиатуре. Для преобразования строки в число надо использовать, например, методы <code>System.ToInt32()</code> или <code>System.ToSingle()</code>
<code>Console.Read()</code>	Значением метода <code>Read</code> является код символа, набранного на клавиатуре

Метод `Console.WriteLine` выводит на экран (в окно консоли) строку, указанную в качестве параметра метода.

Если надо вывести значение числовой переменной, то для преобразования числа в строку следует использовать метод `ToString`. Вид (формат) строки, возвращаемой методом `ToString`, определяет параметр, указанный в инструкции вызова метода (табл. 6.2).

Таблица 6.2. Параметры метода `ToString`

Параметр	Формат	Пример
"C", "c"	<code>Currency</code> — финансовый (денежный). Используется для представления денежных величин. Обозначение денежной единицы, разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	5 5055,28р.
"e", "E"	<code>Scientific (exponential)</code> — научный. Используется для представления очень маленьких или очень больших чисел. Разделитель целой и дробной частей числа задается в настройках операционной системы	5,505528+E04
"f", "F"	<code>Fixed</code> — число с фиксированной точкой. Используется для представления дробных чисел. Количество цифр дробной части, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55055,28

Таблица 6.2 (окончание)

Параметр	Формат	Пример
"g", "G"	General — универсальный формат. Похож на Number, но разряды не разделены на группы	55055,28
"n", "N"	Number — числовой. Используется для представления дробных чисел. Количество цифр дробной части, символ-разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	5 5055,28
"F", "R"	Roundtrip — без округления. В отличие от формата Number, этот формат не выполняет округления (количество цифр дробной части зависит от значения числа)	55055,2755

Следует обратить внимание, что символ-разделитель целой и дробной частей числа задает операционная система. Этот же символ надо использовать и при вводе дробных чисел.

После выполнения инструкции `Write` курсор остается в той позиции экрана, в которую он переместился после вывода последнего символа строки. Следующая инструкция `Write` начинает вывод с той позиции экрана, в которой находится курсор.

Метод `WriteLine` отличается от метода `Write` тем, что после вывода строки, курсор автоматически переходит в начало следующей строки.

Параметр метода `WriteLine` можно не указывать. В этом случае курсор будет переведен в начало следующей строки.

Метод `ReadLine` объекта `Console` обеспечивает ввод с клавиатуры строки символов. Для преобразования введенной строки в данные необходимо использовать соответствующие методы преобразования: `System.ToInt16`, `System.ToInt32`, `System.ToSingle`, `System.ToDouble` и т. д.

Следует обратить внимание на то, что в процессе преобразования строки в число возможны ошибки (исключения), например, из-за того, что при вводе дробного числа пользователь введет точку вместо запятой ("правильным" символом, при стандартной для России настройке операционной системы, является запятая).

В качестве примера использования методов объекта `Console` в листинге 6.1 приведена программа пересчета цены из долларов в рубли.

Листинг 6.1. Консольное приложение

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace usd2rub_con
{
    class Program
```

```
{  
    static void Main(string[] args)  
    {  
        float usd, k, rub;  
  
        string st;  
  
        Console.WriteLine("Конвертер");  
  
        // ввод данных  
        Console.Write("Цена в долларах->");  
        st = Console.ReadLine();  
        usd = Convert.ToSingle(st);  
  
        Console.Write("Курс->");  
        st = Console.ReadLine();  
        k = Convert.ToSingle(st);  
  
        // расчет  
        rub = usd * k;  
  
        // вывод результата  
        Console.WriteLine(); // пустая строка  
  
        st = usd.ToString("f") + "USD = " + rub.ToString("c");  
        Console.WriteLine(st);  
  
        // чтобы окно не исчезло с экрана  
        Console.WriteLine();  
        Console.Write("Press any key to continue...");  
        int ch = Console.Read();  
    }  
}  
}
```

Создание консольного приложения

Для того чтобы создать консольное приложение, надо в меню **File** выбрать команду **New Project**, в появившемся окне **New Project** выбрать **Console Application**, в поле **Name** ввести имя проекта (рис. 6.2) и нажать кнопку **OK**. В результате этих действий станет доступным окно редактора кода, в котором можно набирать текст программы.

В качестве примера в листинге 6.2 приведен текст консольного приложения — утилиты `clear`, которая удаляет из каталога, указанного при запуске утилиты, и всех его подкаталогов tmp-файлы. Также, если при запуске утилиты указан ключ `-CLEARDEBUG`, утилита удаляет все файлы из всех подкаталогов Debug. Так как опе-

рация удаления является потенциально опасной, то утилита запрашивает у пользователя подтверждение на выполнение. Причем, чтобы операция очистки была начата, необходимо ввести Y, ввод любого другого символа, в том числе y, отменяет операцию. Основную работу — удаление файлов выполняет рекурсивная функция clear. Сначала она обрабатывает (удаляет tmp-файлы) текущий каталог, затем формирует список подкаталогов текущего каталога, делает текущим первый (по номеру элементов списка) подкаталог и вызывает себя (рекурсия) для обработки этого подкаталога.

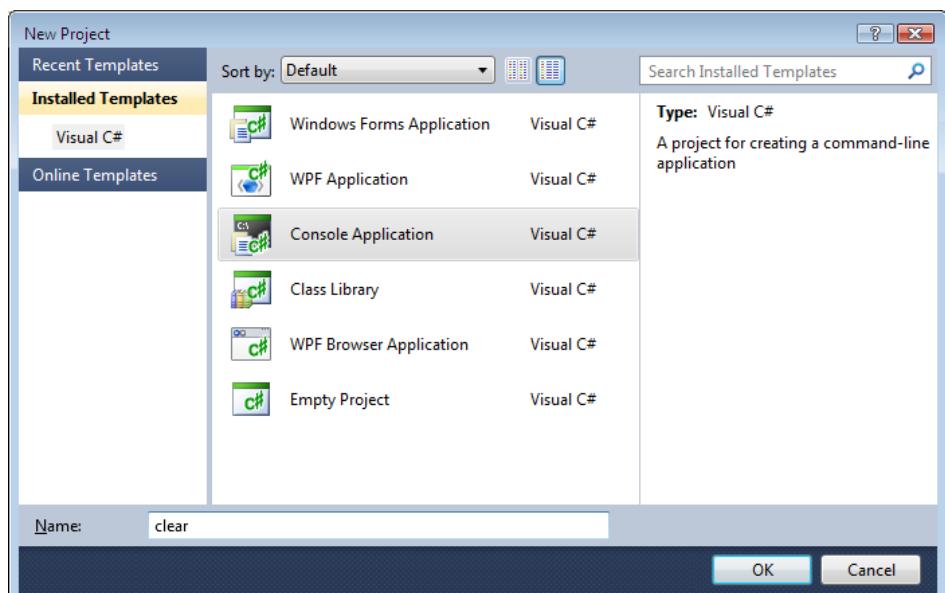


Рис. 6.2. Начало работы над консольным приложением

Листинг 6.2. Очистка диска (консольное приложение)

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.IO;

namespace clear
{
    class Program
    {
        static int nDel = 0;          // количество удаленных файлов
        static bool debug = true;    // true - только сообщение о выполнении
                                     // операции удаления
        static bool clearDebug = false; // очистка каталогов Debug
    }
}

```

```
// Очистка текущего каталога и всех его подкаталогов.  
// Рекурсивная функция  
static int Clear()  
{  
    System.String[] Directories; // подкаталоги текущего каталога  
    System.String[] Files; // файлы, которые надо удалить  
  
    int n;  
    int i;  
  
    if (debug == true)  
        Console.WriteLine("Текущий каталог: " +  
                           System.IO.Directory.GetCurrentDirectory().ToString());  
  
    if (clearDebug)  
    {  
        int p = System.IO.Directory.GetCurrentDirectory().  
                           ToString().LastIndexOf('\\');  
        String aFolder = System.IO.Directory.GetCurrentDirectory().  
                           ToString().Substring(p + 1);  
  
        if (aFolder == "Debug")  
            // удалить все файлы из каталога Debug  
            Files = Directory.GetFiles  
                   (System.IO.Directory.GetCurrentDirectory(), "*.*");  
        else  
            Files = Directory.GetFiles  
                   (System.IO.Directory.GetCurrentDirectory(), "*.tmp");  
    }  
    else  
    {  
        // удалить tmp-файлы из текущего каталога  
        Files = Directory.GetFiles  
                   (System.IO.Directory.GetCurrentDirectory(), "*.tmp");  
    }  
  
    int nf = Files.Length;  
    for (i = 0; i < nf; i++)  
    {  
        if (!debug)  
            // УДАЛЕНИЕ ФАЙЛОВ  
            System.IO.File.Delete(Files[i]);  
  
        Console.WriteLine(Files[i] + " - удален");  
        nDel = nDel + 1;  
    }  
}
```

```
// обработать подкаталоги
if (debug)
    Console.WriteLine("Подкаталоги:");

Directories =
    Directory.GetDirectories(Directory.GetCurrentDirectory());
n = Directories.Length;
for (i = 0; i < n; i++)
{
    if (debug)
        Console.WriteLine(Directories[i]);
    Directory.SetCurrentDirectory(Directories[i]);
    Clear();
};

if (debug)
    Console.WriteLine("Обработан!");

return 0;
}

static int Main(string[] args)
{
    String path;

Console.WriteLine("Очистка диска");

// проверим, указан ли каталог
if (args.Length == 0)
{
    Console.WriteLine("Ошибка: надо задать каталог.");
    Console.WriteLine("Команда: clear <Каталог> [-CLEARDEBUG]");
    Console.WriteLine("-CLEARDEBUG - очистка каталогов Debug\n");

    Console.Write("Нажмите <Enter>");
    Console.ReadLine();
    return -1;
}

path = args[0];

if ((args.Length >1) && (args[1] == "-CLEARDEBUG"))
    clearDebug = true;

// проверим, существует ли указанный каталог
try
```

```
{  
    Directory.SetCurrentDirectory(path);  
}  
catch (System.IO.DirectoryNotFoundException ex)  
{  
    Console.WriteLine("Каталог " + path + " не найден");  
    //Console.WriteLine(e.Message);  
    Console.Write("Нажмите <Enter>");  
    Console.ReadLine();  
    return -1;  
};  
  
if ( clearDebug)  
{  
    Console.WriteLine  
        ("Программа удалит все файлы из подкаталогов Debug");  
    Console.WriteLine  
        ("кataloga "+ path+ " и всех его подкаталогов.");  
}  
else  
{  
    Console.WriteLine("Программа удалит tmp-файлы");  
    Console.WriteLine("из каталога "+ path+  
        " и всех его подкаталогов.");  
}  
  
Console.Write("Выполнить? (Y/N) ");  
string r = Console.ReadLine();  
  
if (r != "Y")  
{  
    Console.Write("Отмена операции\nНажмите <Enter>");  
    Console.ReadLine();  
    return -1;  
}  
  
if (debug)  
    Console.WriteLine("Отладка (только сообщения об удалении)");  
  
int res = Clear();  
  
Console.WriteLine("Удалено файлов: "+ nDel.ToString());  
Console.WriteLine("Работа выполнена!");  
Console.Write("Нажмите <Enter>");  
Console.ReadLine();  
    return 0;  
}  
}  
}
```

Построение консольного приложения выполняется обычным образом, т. е. выбором в меню **Build** команды **Build Solution**.

После успешной компиляции программу можно запустить. Для этого надо в меню **Debug** выбрать команду **Start Debugging** или **Start Without Debugging**.

Если программа должна получать параметры из командной строки, то при ее запуске из среды разработки параметры надо ввести в поле **Command line arguments** вкладки **Debug** окна **Property Page** (рис. 6.3), которое становится доступным в результате выбора в меню **Project** команды **Properties**. Следует обратить внимание: если в строке-параметре есть пробелы, то ее надо заключить в двойные кавычки.

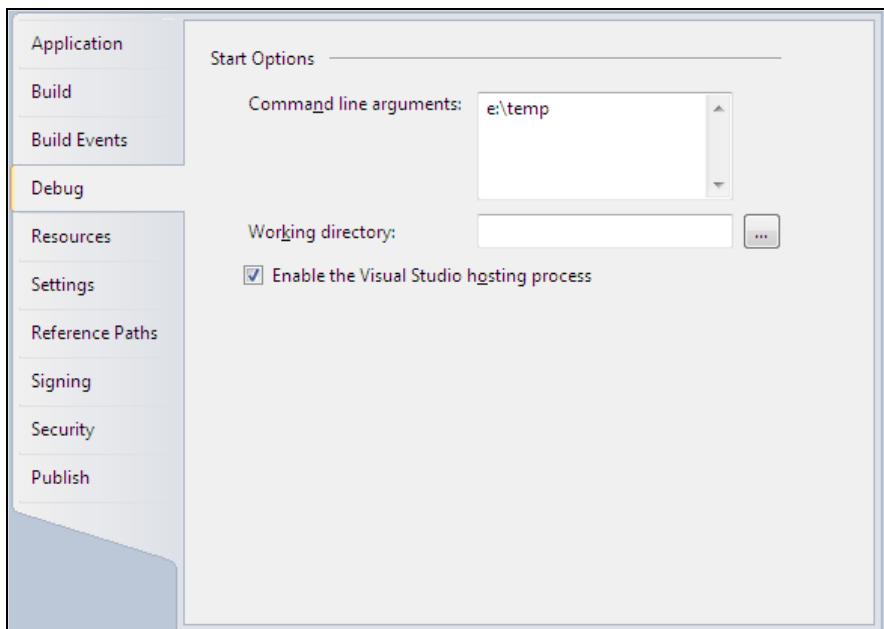
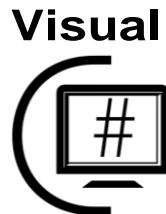


Рис. 6.3. При запуске программы из Visual C# параметры командной строки надо ввести в поле **Command line arguments**

Запуск консольного приложения

Из операционной системы консольное приложение можно запустить, сделав двойной щелчок на его имени. Если при запуске необходимо указать параметры, то надо открыть окно командной строки (оно становится доступным в результате выбора в меню **Пуск** команды **Все программы** ▶ **Стандартные** ▶ **Командная строка**) и набрать команду запуска там. Команду запуска консольного приложения можно набрать также в окне **Выполнить**, которое становится доступным в результате выбора в меню **Пуск** команды **Все программы** ▶ **Выполнить**.



ГЛАВА 7

LINQ

Технология LINQ (Language Integrated Query, интегрированный язык запросов) впервые была представлена в C# 3.0 и .NET Framework 3.5.

Концепция LINQ заключается в использовании для доступа к данным и манипулирования ими специального языка запросов, интегрированного в язык программирования. При этом в качестве источника данных может выступать массив, список, XML-документ или даже таблица базы данных.

Лямбда-выражение

Лямбда-выражение — сокращенный способ объявления анонимной функции (делегата). Например, лямбда-выражение `st => st.ToUpper()` трактуется так: взять строку `st` и вернуть в качестве результата строку `st`, преобразованную к верхнему регистру. Выражение `st => st.Contains("Культин")` трактуется так: взять строку `st` и вернуть в качестве результата `True`, если в строке `st` есть подстрока `Культин`, или `False`, если подстроки `Культин` в строке `st` нет.

Лямбда-выражения используются в качестве параметра многих Q-операторов (см. далее).

Q-оператор

Выполнение операций над данными обеспечивают Query-операторы (Q-операторы). Q-оператор (метод) выполняет действие над последовательностью (sequence) данных, преобразует исходную последовательность (набор) данных в результат. Например, значением оператора `First` является первый элемент последовательности, а оператора `Where` — последовательность, образованная из элементов обрабатываемой последовательности, удовлетворяющих условию, заданному параметром оператора.

Некоторые Q-операторы и примеры их использования (`names` — массив строк, `price` — дробный массив) приведены в табл. 7.1.

Таблица 7.1. Некоторые Q-операторы

Q-оператор	Действие	Тип	Примеры
Where (<i>Lambda</i>)	Формирует последовательность путем отбора элементов. Критерий отбора задает <i>Lambda</i> -выражение	Последовательность	names.Where(n => n.Contains("Культин")) price.Where(p => p > 1500)
Select (<i>Lambda</i>)	Формирует последовательность путем заданной <i>Lambda</i> -выражением обработки исходной последовательности	Последовательность	names.Select(st => st.ToUpper()) price.Select(p => p*0.85)
Order (<i>Lambda</i>)	Формирует упорядоченную по возрастанию последовательность. Выражение <i>Lambda</i> задает правило сравнения элементов	Последовательность	price.OrderBy(p => p)
Reverse ()	Изменяет порядок следования элементов: первый элемент меняется местом с последним, второй — с предпоследним, и т. д.	Последовательность	Сортировка по убыванию: nprice = price.Order(p => p); nprice = nprice.Reverse();
First()	Первый элемент последовательности	Элемент	names.Firstst()
First (<i>Lambda</i>)	Первый элемент последовательности, удовлетворяющий условию, заданному выражением <i>Lambda</i>	Элемент	names.Firstst(n => n.Contains("Культин"))
Last()	Последний элемент последовательности	Элемент	names.Last()
Last (<i>Lambda</i>)	Последний элемент последовательности, удовлетворяющий условию, заданному выражением <i>Lambda</i>	Элемент	names.Last(n => n.Contains("Культин"))
ElementAt (<i>n</i>)	Возвращает элемент с указанным номером	Элемент	names.ElementAt(3)
Contains (<i>value</i>)	Возвращает True, если в последовательности есть элемент с указанным значением	bool	names.Contains("Никита Культин") price.Contains(1020)
Min()	Возвращает минимальный элемент последовательности	Элемент	price.Min();

Таблица 7.1 (окончание)

Q-оператор	Действие	Тип	Примеры
Max ()	Возвращает максимальный элемент последовательности	Элемент	price.Max ()
Average ()	Возвращает среднее арифметическое значений элементов последовательности	double	price.Average ()
Sum ()	Вычисляет сумму значений элементов последовательности		price.Sum ()
Sum (<i>Lambda</i>)	Преобразует в соответствии с выражением <i>Lambda</i> и вычисляет сумму значений элементов последовательности		price.Sum (p => p/100))
Count ()	Количество элементов последовательности	Целое	price.Count ()
Any (<i>Lambda</i>)	Возвращает True, если в последовательности есть элемент, удовлетворяющий условию <i>Lambda</i>	bool	price.Any (p => p < 500)

Выполнение Q-оператора

Q-оператор выполняет действие над данными, преобразует исходный набор данных, например массив, в результат — последовательность.

Инструкция вызова Q-оператора в общем виде выглядит так:

```
IEnumerable<T> s2 = System.Linq.Enumerable.Q(s1, Lambda);
```

где:

- ◆ *Q* — Q-оператор;
- ◆ *s1* — данные (последовательность), над которыми выполняется действие;
- ◆ *Lambda* — лямбда-выражение, параметр Q-оператора;
- ◆ *s2* — последовательность, результат выполнения Q-оператора;
- ◆ *T* — тип элементов последовательности-результата.

Пример:

```
string[] names = { "Никита Кульгин", "Платон Кульгин",
                   "Данила Кульгин", "Лариса Цой", };
IEnumerable<string> s =
    System.Linq.Enumerable.Where(names, n => n.Contains("Кульгин"));
```

Так как Q-операторы реализованы как методы, то их можно применять напрямую к объектам. То есть приведенную инструкцию вызова Q-оператора Where можно записать так:

```
IEnumerable<string> s = names.Where(n => n.Contains("Кульгин"));
```

Операции с массивами

Как было сказано ранее, технология LINQ позволяет выполнять операции над различными источниками данных, и в частности над массивами.

Поиск в массиве

Программа "LINQ-поиск в массиве" (листинг 7.1), ее форма и окно (результат поиска) приведены на рис. 7.1 и 7.2, демонстрируют использование LINQ для поиска информации в массиве. Выбор информации обеспечивает оператор Where.

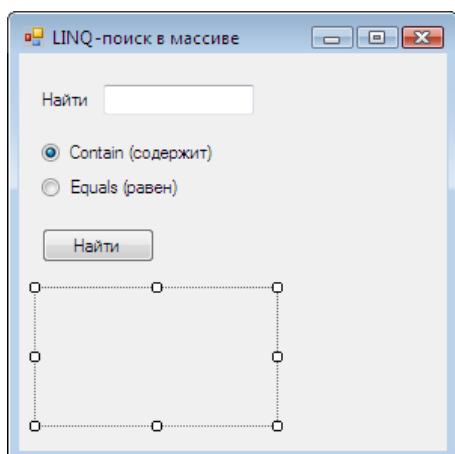


Рис. 7.1. Форма программы
"LINQ-поиск в массиве"

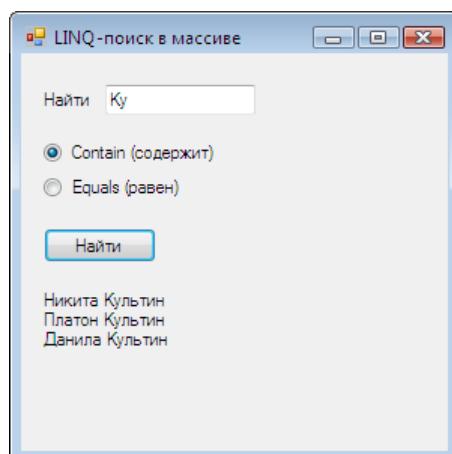


Рис. 7.2. Окно программы
"LINQ-поиск в массиве"

Листинг 7.1. LINQ-поиск в массиве

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
```

```
{  
    // массив  
    string[] names = { "Никита Кульгин", "Платон Кульгин",  
                       "Данила Кульгин", "Лариса Цой", };  
  
    public Form1()  
    {  
        InitializeComponent();  
    }  
  
    // щелчок на кнопке Найти  
    private void button1_Click(object sender, EventArgs e)  
    {  
        // образец для поиска  
        string aName;  
  
        // результат поиска  
        IEnumerable<string> filteredNames;  
  
        aName = textBox1.Text;  
  
        if (radioButton1.Checked)  
        {  
            // поиск подстроки  
            filteredNames = System.Linq.Enumerable.Where  
                (names, n => n.Contains(aName));  
        }  
        else  
        {  
            // равенство  
            filteredNames = System.Linq.Enumerable.Where  
                (names, n => n.Equals(aName));  
        }  
  
        label1.Text = "";  
  
        foreach (string st in filteredNames)  
            label1.Text = label1.Text + st + '\n';  
    }  
}
```

Обработка массива

Программа "LINQ-обработка массива" (листинг 7.2), ее окно приведено на рис. 7.3, демонстрирует использование Q-операторов для поиска минимального и максимального элементов, вычисления суммы элементов, а также для выполнения сортировки.

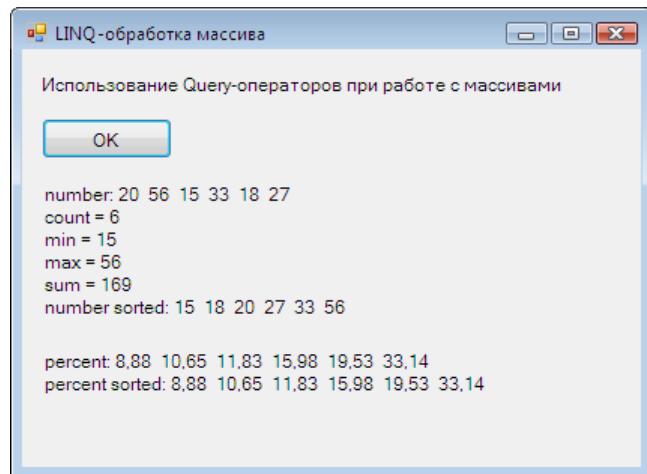


Рис. 7.3. Окно программы "LINQ-обработка массива"

Листинг 7.2. LINQ-обработка массива

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // массив
        double[] numbers = { 20, 56, 15, 33, 18, 27 };

        private void button1_Click(object sender, EventArgs e)
        {
            double count, min, max, sum;

            count = numbers.Count(); // количество элементов
            max = numbers.Max(); // максимальный элемент
        }
    }
}

```

```
min = numbers.Min();           // минимальный элемент
sum = numbers.Sum();          // сумма элементов массива

label2.Text = "number: ";
for (int i = 0; i < count; i++)
{
    label2.Text = label2.Text + numbers[i].ToString() + " ";
}

label2.Text = label2.Text + "\ncount = " + count.ToString() +
             "\nmin = " + min.ToString() +
             "\nmax = " + max.ToString() +
             "\nsum = " + sum.ToString();

// список sorted содержит упорядоченные
// по возрастанию элементы массива numbers
IEnumerable<double> sorted = numbers.OrderBy(n => n);

label2.Text = label2.Text + "\nnumber sorted: ";
foreach (double a in sorted)
    label2.Text = label2.Text + a.ToString() + " ";

// элемент percent(i) – доля i-го элемента sorted[i]
// в общей сумме элементов массива sorted
IEnumerable<double> percent = numbers.Select(n => n/sum*100);

percent = percent.OrderBy(n => n);

label3.Text = "percent: ";
foreach (double a in percent)
    label3.Text = label3.Text + a.ToString("f") + " ";

label3.Text = label3.Text + "\npercent sorted: ";
foreach (double a in percent)
    label3.Text = label3.Text + a.ToString("f") + " ";
}

}
```

Обработка массива записей

Технология LINQ позволяет выполнять операции над массивами различного типа. В предыдущих примерах обрабатывались строковые и числовые массивы. Программа "LINQ-обработка массива записей" (листинг 7.3), ее окно приведено на рис. 7.4, показывает использование Q-операторов для обработки массива записей.

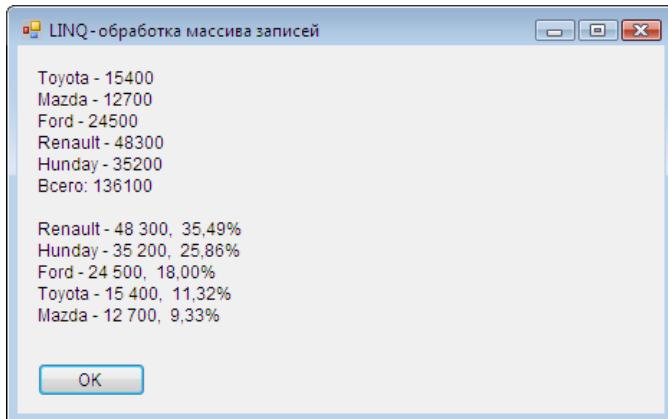


Рис. 7.4. Окно программы "LINQ-обработка массива записей"

Листинг 7.3. LINQ-обработка массива записей

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        class sale
        {
            public string Title;
            public int n;
            public double p;
        }

        sale[] sales = { new sale { Title = "Toyota", n = 15400 },
                        new sale { Title = "Mazda", n = 12700 },
                        new sale { Title = "Ford", n = 24500 },
                        new sale { Title = "Renault", n = 48300 },
                        new sale { Title = "Hundai", n = 35200 }
                    };

        public Form1()
        {
            InitializeComponent();
        }
    }
}

```

```
private void button1_Click(object sender, EventArgs e)
{
    // отобразить исходные данные
    string st = "";
    foreach (sale aSale in sales)
        st = st + aSale.Title + " - " + aSale.n.ToString() + "\n";

    // обработка данных
    long sum = 0;
    // IEnumerable<int> count = sales.Select(a => a.n);
    // sum = count.Sum();

    foreach (sale aSale in sales)
        sum = sum + aSale.n;

    st = st + "Всего: " + sum.ToString();

    // вычислить долю каждой категории в общей сумме
    foreach (sale aSale in sales)
        aSale.p = (double)aSale.n / sum;

    // сортировка:
    // OrderBy - по возрастанию
    // OrderByDescending - по убыванию
    st = st + "\n\n";
    IEnumerable<sale> sorted = sales.OrderByDescending(a => a.n);

    // вывод результата
    foreach (sale aSale in sorted)
        st = st + aSale.Title + " - " +
            aSale.n.ToString("### 000") + ", " +
            (aSale.p * 100).ToString("f") + "%\n";

    label1.Text = st;
}
}
```

Работа с XML-документами

Отображение XML-документа

Программа "LINQ-отображение XML-документа" (ее окно приведено на рис. 7.5, текст — в листинге 7.4) показывает, как, используя LINQ, можно прочитать и отобразить информацию, находящуюся в XML-файле.

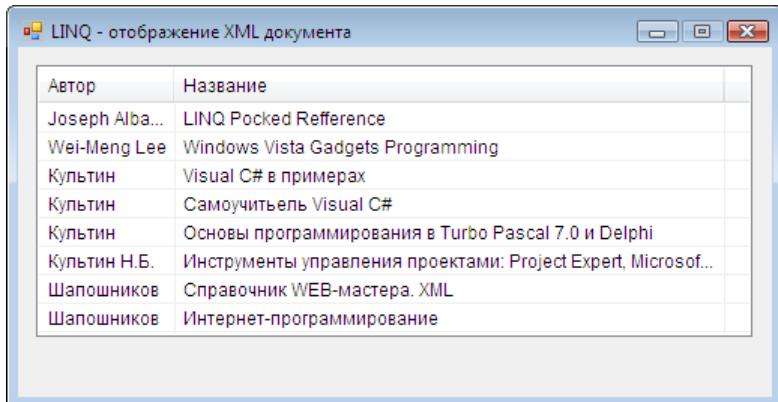


Рис. 7.5. Окно программы "LINQ-отображение XML-документа"

Листинг 7.4. LINQ-отображение XML-документа

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// эти ссылки вставлены вручную
using System.Xml.Linq;
using System.IO;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        XDocument doc;
        DirectoryInfo di;

        public Form1()
        {
            InitializeComponent();

            // настройка компонента listView
            listView1.View = View.Details;
            listView1.GridLines = true;
            //listView1.Sorting = SortOrder.Ascending;
        }
    }
}

```

```
listView1.Columns.Add("Автор");
listView1.Columns[0].Width = 90;
listView1.Columns.Add("Название");
listView1.Columns[1].Width = listView1.Width -
                           listView1.Columns[0].Width - 4 - 17;
// 4 - ширина границы; 17 - ширина полосы прокрутки

// предполагается, что XML-файл находится в каталоге ApplicationData
di = new DirectoryInfo(Environment.GetFolderPath
    (Environment.SpecialFolder.ApplicationData));

// загрузить данные из XML-файла
try
{
    doc = XDocument.Load(di.FullName + "\\books.xml");
}
catch (Exception aException)
{
    MessageBox.Show(aException.Message, "Load XML",
                   MessageBoxButtons.OK, MessageBoxIcon.Error);
    return;
}

// отобразить данные
IEnumerable< XElement> query = from b in doc.Elements().Elements()
                                    select b;

listView1.Items.Clear();

// * вывести список элементов (узлов) XML-документа *
// Чтобы добавить в компонент ListView строку,
// точнее, ячейку в первый столбец,
// надо добавить элемент в коллекцию Items.
// Чтобы данные появились во второй строке,
// надо добавить подэлемент (SubItem) в коллекцию SubItems,
// соответствующую той строке, в которую надо добавить ячейку.
foreach ( XElement e1 in query)
{
    bool newElement = true;
    foreach ( XElement e2 in e1.Elements())
    {
        if (newElement == true)
        {
            // первый узел - элемент
            listView1.Items.Add(e2.Value);
            newElement = false;
        }
    }
}
```

```
        else
        {
            // остальные узлы - подэлементы
            listView1
                .Items[listView1.Items.Count - 1]
                .SubItems.Add(e2.Value);
        }
    }
}
```

Программа "LINQ to XML" (листинг 7.5), ее форма приведена на рис. 7.6, показывает, как можно, используя LINQ, создать XML-документ, загрузить существующий документ и добавить в него информацию. Кроме этого, программа показывает, как выбрать из XML-документа нужную информацию и отобразить ее в поле компонента `ListView`.

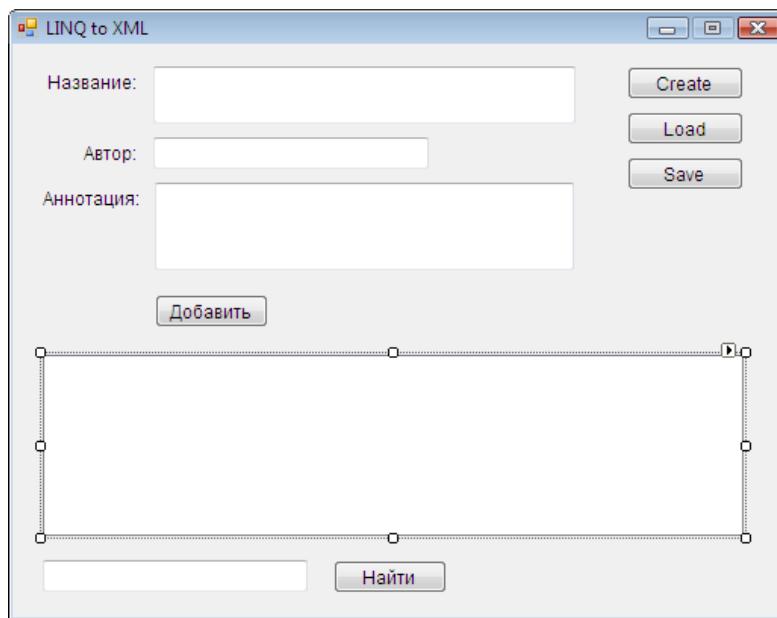


Рис. 7.6. Форма программы "LINQ to XML"

Листинг 7.5. LINQ-работа с XML-документом

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// эти ссылки вставлены вручную
using System.Xml.Linq;
using System.IO;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        XDocument doc;
        DirectoryInfo di;

        public Form1()
        {
            InitializeComponent();
            di = new DirectoryInfo(Environment.GetFolderPath
                (Environment.SpecialFolder.ApplicationData));

            // настройка компонента listView
            listView1.View = View.Details;
            listView1.GridLines = true;
            //listView1.Sorting = SortOrder.Ascending;

            listView1.Columns.Add("Автор");
            listView1.Columns[0].Width = 90;
            listView1.Columns.Add("Название");
            listView1.Columns[1].Width =
                listView1.Width - listView1.Columns[0].Width - 4;
        }

        // загрузить XML-документ из файла
        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                doc = XDocument.Load(di.FullName + "\\books.xml");
                button1.Enabled = true;
                button4.Enabled = true;
                button5.Enabled = true;
            }
            catch (Exception aException)
            {
                MessageBox.Show(aException.Message, "Load XML",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
}
```

```
        return;
    }

    IEnumerable< XElement> books = doc.Elements();
    foreach ( XElement aBooks in books.Elements())
    {
        listView1.Items.Add(aBooks.Element("Author").Value);
        listView1.Items[listView1.Items.Count - 1].SubItems.
            Add(aBooks.Element("Title").Value);
    }
    button2.Enabled = false;
}

// добавить элемент в XML-документ
private void button1_Click(object sender, EventArgs e)
{
    doc.Element("books").Add(new XElement("book",
        new XElement("Author", textBox2.Text),
        new XElement("Title", textBox1.Text),
        new XElement("Description", textBox4.Text))
    );
}

// создать XML-файл
private void button2_Click(object sender, EventArgs e)
{
    FileInfo fi = new FileInfo(di.FullName+"\\"+books.xml");
    if (fi.Exists)
    {
        DialogResult dr;
        dr = MessageBox.Show("Файл " + fi.FullName +
            " существует.\nЗаменить его новым?", "Create XML",
            MessageBoxButtons.YesNo, MessageBoxIcon.Warning,
            MessageBoxDefaultButton.Button2);
        if (dr == DialogResult.No)
        {
            return;
        }
    }
}

doc =
new XDocument(
    new XElement("books",
        new XElement("book",
            new XElement("Author", "Кульгин Н.Б."),
            new XElement("Title", "Visual C# в примерах"),
            new XElement("Description", ""))),
```

```
new XElement("book",
    new XElement("Author", "Н. Культин"),
    new XElement("Title", "Самоучитель Visual C#"),
    new XElement("Description", "")),
    new XElement("book",
        new XElement("Author", "Культин"),
        new XElement("Title", "Turbo Pascal 7.0 и Delphi"),
        new XElement("Description", ""))
)
);
doc.Save(di.FullName + "\\books.xml");
}

// сохранить изменения
private void button4_Click(object sender, EventArgs e)
{
    doc.Save(di.FullName + "\\books.xml");
}

// щелчок на кнопке Найти
private void button5_Click(object sender, EventArgs e)
{
    // выбрать элементы, у которых поле Author
    // содержит текст, введенный пользователем
    IEnumerable< XElement> query =
        from b in doc.Elements().Elements()
        where b.Elements("Author").Any
            (n => n.Value.Contains(textBox3.Text))
        select b;

    listView1.Items.Clear();

    // вывести список элементов
    foreach ( XElement e1 in query)
    {
        //st = st + "\n" + xe.ToString();
        bool firstElement = true;
        foreach ( XElement e2 in e1.Elements())
        {
            if (firstElement == true)
            {
                listView1.Items.Add(e2.Value);
                firstElement = false;
            }
        }
    }
}
```

```
    else
    {
        listView1.Items[listView1.Items.Count-1]
            .SubItems.Add(e2.Value);
    }
}
}
}
```

ГЛАВА 8

Отладка программы

Успешное завершение процесса построения не означает, что в программе нет ошибок. Убедиться, что программа работает правильно, можно только в процессе проверки ее работоспособности, который называется *тестированием*.

Обычно программа редко сразу начинает работать так, как надо, или работает правильно только на некотором ограниченном наборе исходных данных. Это свидетельствует о том, что в программе есть алгоритмические ошибки. Процесс поиска и устранение ошибок называется *отладкой*.

Классификация ошибок

Ошибки, которые могут быть в программе, принято делить на три группы:

- ◆ синтаксические;
- ◆ ошибки времени выполнения;
- ◆ алгоритмические.

Синтаксические ошибки, их также называют *ошибками времени компиляции* (Compile-time error), наиболее легко устранимы. Их обнаруживает компилятор, а программисту остается только внести изменения в текст программы и выполнить повторную компиляцию.

Ошибки времени выполнения, они называются исключениями (exception), тоже, как правило, легко устранимы. Они обычно проявляются уже при первых запусках программы и во время тестирования.

При возникновении исключения в программе, запущенной из среды разработки, на экране появляется окно, в котором отображаются сведения о типе исключения и информационное сообщение, поясняющее причину его возникновения. На рис. 8.1 приведен пример сообщения об исключении `FileNotFoundException`, причина которого — отсутствие (недоступность) файла, нужного программе.

Если программа запущена из операционной системы, то при возникновении исключения на экране также появляется сообщение об ошибке (рис. 8.2). Сделав в этом окне щелчок на кнопке **Показать подробности проблемы**, можно увидеть тип исключения (рис. 8.3).

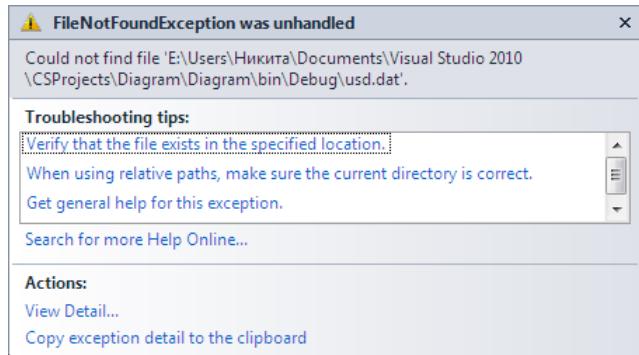


Рис. 8.1. Пример сообщения об ошибке (программа запущена из Microsoft Visual C#)

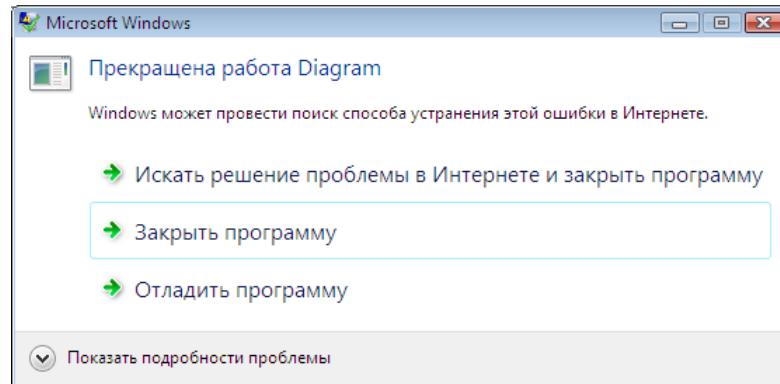


Рис. 8.2. Сообщение об ошибке при запуске программы из операционной системы

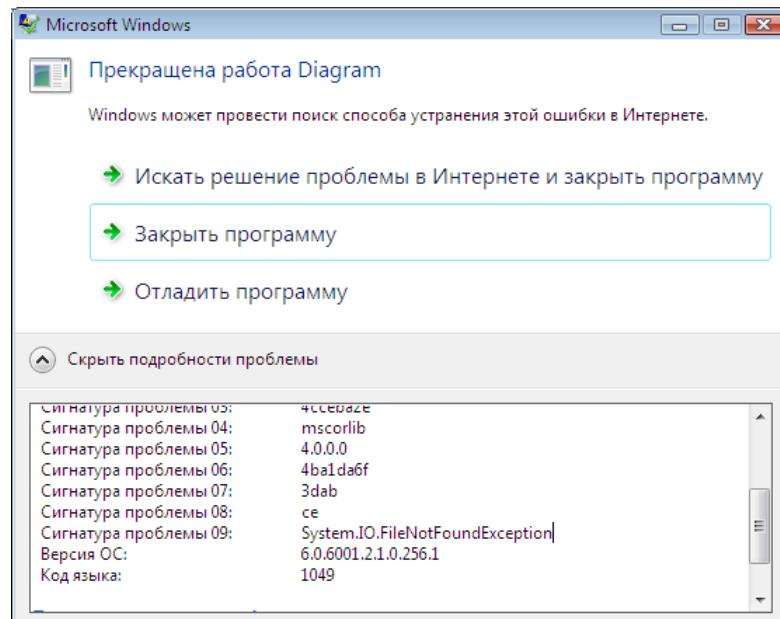


Рис. 8.3. Подробная информация об исключении при запуске программы из операционной системы

С алгоритмическими ошибками дело обстоит иначе. Компилятор обнаружить их не может. Поэтому даже в том случае, если в программе есть алгоритмические ошибки, компиляция завершается успешно. Убедиться в том, что программа работает правильно и в ней нет алгоритмических ошибок, можно только в процессе **тестирования** программы. Если программа работает не так как надо, а результат ее работы не соответствует ожидаемому, то, скорее всего, в ней есть алгоритмические ошибки. Процесс поиска алгоритмических ошибок может быть достаточно трудоемким. Чтобы найти алгоритмическую ошибку, программисту приходится анализировать алгоритм, вручную "прокручивать" процесс его выполнения.

Предотвращение и обработка ошибок

Как было сказано ранее, в программе во время ее работы могут возникать исключения (ошибки). Наиболее часто исключения возникают вследствие действий пользователя. Он, например, может ввести неверные данные или, что бывает довольно часто, удалить нужный программе файл.

Нарушение в работе программы называется *исключением*. Стандартную обработку исключений, которая в общем случае заключается в отображении сообщения об ошибке, берет на себя автоматически добавляемый в выполняемую программу код. Вместе с тем программист может (и должен) обеспечить явную обработку исключений. Для этого в текст программы необходимо поместить инструкции, обеспечивающие обработку возможных исключений.

Инструкция обработки исключения в общем виде выглядит так:

```
try
{
    // здесь инструкции, выполнение которых может вызвать исключение
}

// начало секции обработки исключений
catch (ExceptionType_1 e)
{
    // инструкции обработки исключения ExceptionType_1
}
catch (ExceptionType_2 e)
{
    // инструкции обработки исключения ExceptionType_2
}

catch (ExceptionType_k e)
{
    // инструкции обработки исключения ExceptionType_k
}
```

Здесь:

- ◆ **try** — ключевое слово, показывающее, что далее следуют инструкции, при выполнении которых возможно возникновение исключений, и что обработку этих исключений берет на себя программа;

- ◆ `catch` — ключевое слово, обозначающее начало секции обработки исключения указанного типа. Инструкции этой секции будут выполнены при возникновении исключения указанного типа;
- ◆ `ExceptionType_i` — тип исключения.

Следует обратить внимание, после обработки исключения, инструкции секции `try`, следующие за вызвавшей исключение инструкцией, не выполняются.

В качестве примера в листинге 8.1 приведен конструктор формы программы "Диаграмма", в который включены инструкции, обеспечивающие обработку наиболее вероятных исключений: `FileNotFoundException` (Не найден файл) и `FormatException` (Неверный формат данных). Первое исключение возникает, если недоступен файл, второе — если в файле неверные данные (например, дробные числа записаны с использованием точки). Обработка исключения заключается в выводе сообщения о возникновении ошибки.

Листинг 8.1. Пример обработки исключений

```
public Form1()
{
    InitializeComponent();

    // чтение данных из файла в массив
    System.IO.StreamReader sr; // поток для чтения
    try
    {
        // Создаем поток для чтения.
        // Application.StartupPath возвращает путь
        // к каталогу, из которого была запущена программа
        sr = new System.IO.StreamReader(
            Application.StartupPath + "\\usd.dat");

        // создаем массив
        d = new double[10];

        // читаем данные из файла
        int i = 0;
        string t = sr.ReadLine();
        while ((t != null) && (i < d.Length))
        {
            // записываем считанное число в массив
            d[i++] = Convert.ToDouble(t);
            t = sr.ReadLine();
        }

        // закрываем поток
        sr.Close();
    }
}
```

```

// задаем функцию обработки события Paint
this.Paint += new PaintEventHandler(drawDiagram);
}

// обработка исключений:

// файл данных не найден
catch (System.IO.FileNotFoundException ex)
{
    MessageBox.Show(ex.Message + "\n" +
        "(" + ex.GetType().ToString() + ")",
        "График",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

// неверный формат данных
catch (System.FormatException ex)
{
    MessageBox.Show("Неверный формат данных.\n(" +
        ex.Message,
        "График",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

```

Основной характеристикой исключения является его тип. В табл. 8.1 перечислены наиболее часто возникающие исключения и указаны причины, которые могут привести к их возникновению.

Таблица 8.1. Типичные исключения

Тип исключения	Исключение, причина
SystemFormatException	Ошибка формата. Возникает при выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому виду. Наиболее часто возникает при попытке преобразовать строку символов в число, если строка содержит неверные символы. Например, при преобразовании строки в дробное значение, если в качестве десятичного разделителя вместо запятой (при стандартной для России настройке ОС) поставлена точка
System.IndexOutOfRangeException	Выход значения индекса за границу области допустимого значения. Возникает при обращении к несуществующему элементу массива
System.IO.FileNotFoundException	Нет файла. Причина — отсутствие требуемого файла в указанном каталоге
System.IO.DirectoryNotFoundException	Нет каталога. Причина — отсутствие требуемого каталога

Отладчик

Среда разработки предоставляет программисту мощное средство поиска и устранения ошибок в программе — отладчик. Отладчик позволяет выполнять *трассировку* программы, наблюдать значения переменных, контролировать данные.

Трассировка программы

Во время работы программы ее инструкции выполняются одна за другой со скоростью работы процессора компьютера. При этом программист не может определить, какая инструкция выполняется в данный момент, и, следовательно, выяснить, соответствует ли реальный порядок выполнения инструкций разработанному им алгоритму.

В случае неправильной работы программы необходимо видеть реальный порядок выполнения инструкций. Это можно сделать, выполнив трассировку программы. *Трассировка* — это процесс выполнения программы по шагам (step-by-step), инструкция за инструкцией. Во время трассировки программист дает команду: выполнить очередную инструкцию программы.

Существуют два режима трассировки: без захода в процедуру (**Step Over**) и с заходом в процедуру (**Step Into**). Режим трассировки без захода в процедуру производит трассировку выбранной функции, при этом трассировка подпрограмм не осуществляется, вся подпрограмма (функция) выполняется за один шаг. В режиме трассировки с заходом в процедуру (функцию) осуществляется трассировка всей программы, т. е. по шагам выполняется не только главная программа, но и все подпрограммы.

Для того чтобы начать трассировку, необходимо в меню **Debug** выбрать команду **Step Over**. В результате в окне редактора кода будет выделена первая выполняемая инструкция программы. Для того чтобы эта инструкция была выполнена, надо еще раз в меню **Debug** выбрать команду **Step Over** или нажать клавишу **<F10>**. После выполнения текущей инструкции будет выделена следующая. Таким образом, нажимая клавишу **<F10>**, можно видеть процесс выполнения программы.

В любой момент времени можно завершить трассировку и продолжить выполнение программы в реальном темпе. Для этого в меню **Debug** надо выбрать команду **Continue** или нажать клавишу **<F5>**.

Во время трассировки можно наблюдать не только порядок выполнения инструкций программы, но и значения переменных. О том, как это сделать, рассказывается в одном из следующих разделов.

Точки останова программы

При отладке широко используется метод, который называют *методом точек останова*. Суть метода заключается в том, что программист помечает некоторые инструкции программы (ставит точки останова), при достижении которых программа приостанавливает свою работу, и программист может начать трассировку или проконтролировать значения переменных.

Добавление точки останова

Для того чтобы поставить в программу *точку останова* (breakpoint), нужно в окне редактора кода установить курсор в ту строку программы, в которой находится инструкция, перед которой надо поставить точку останова, и в меню **Debug** выбрать команду **Toggle Breakpoint**. В результате в программу будет добавлена точка останова, а инструкция, перед которой точка поставлена, будет отмечена красной точкой (рис. 8.4).

The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The tab bar at the top has 'Program.cs', 'Form1.cs X', and 'Form1.cs [Design]'. The 'Form1.cs' tab is active. The code editor displays C# code for a Windows application. A red circular breakpoint icon is positioned to the left of the first line of code. The code itself is as follows:

```

* ClientSize - размер внутренней области окна
*
* график строим в отклонениях от минимального
* значения ряда данных, так, чтобы он занимал
* всю область построения
*/
double max = d[0]; // максимальный эл-т массива
double min = d[0]; // минимальный эл-т массива

for (int i = 1; i < d.Length; i++)
{
    if (d[i] > max) max = d[i];
    if (d[i] < min) min = d[i];
}

// рисуем диаграмму
int x1, y1; // координаты левого верхнего угла столбика
int w, h; // размер столбца

// ширина столбиков диаграммы
// 5 - расстояние между столбиками

```

Рис. 8.4. Перед инструкцией `max = d[0]` поставлена точка останова

Точку останова можно добавить так же, нажав клавишу **<F9>** или сделав щелчок левой кнопкой мыши в области отображения точек останова. Повторное нажатие клавиши **<F9>** или щелчок на значке точки останова удаляет поставленную точку останова.

Удаление точки останова

Для того чтобы удалить точку останова, нужно в окне редактора кода установить курсор в ту строку программы, в которой находится точка останова, и нажать клавишу **<F9>** (в меню **Debug** выбрать команду **Toggle Breakpoint**) или сделать щелчок левой кнопкой мыши на значке точки останова.

Чтобы удалить все точки останова, надо в меню **Debug** выбрать команду **Delete All Breakpoints**. Следует обратить внимание на то, что программист может, не удаляя точки останова, приостановить их действия. Для этого в меню **Debug** надо выбрать команду **Disable All Breakpoints**.

Наблюдение значений переменных

Во время отладки, при выполнении программы по шагам, бывает полезно знать, чему равно значение той или иной переменной. Отладчик позволяет наблюдать значения переменных программы.

При достижении точки останова в нижней части экрана, на вкладке **Locals**, отображаются значения переменных программы (рис. 8.5). В приведенном примере красная точка показывает точку останова, стрелка — инструкцию, на которой приостановлено выполнение программы в данный момент. Текущие значения переменных отображаются на вкладке **Locals**.

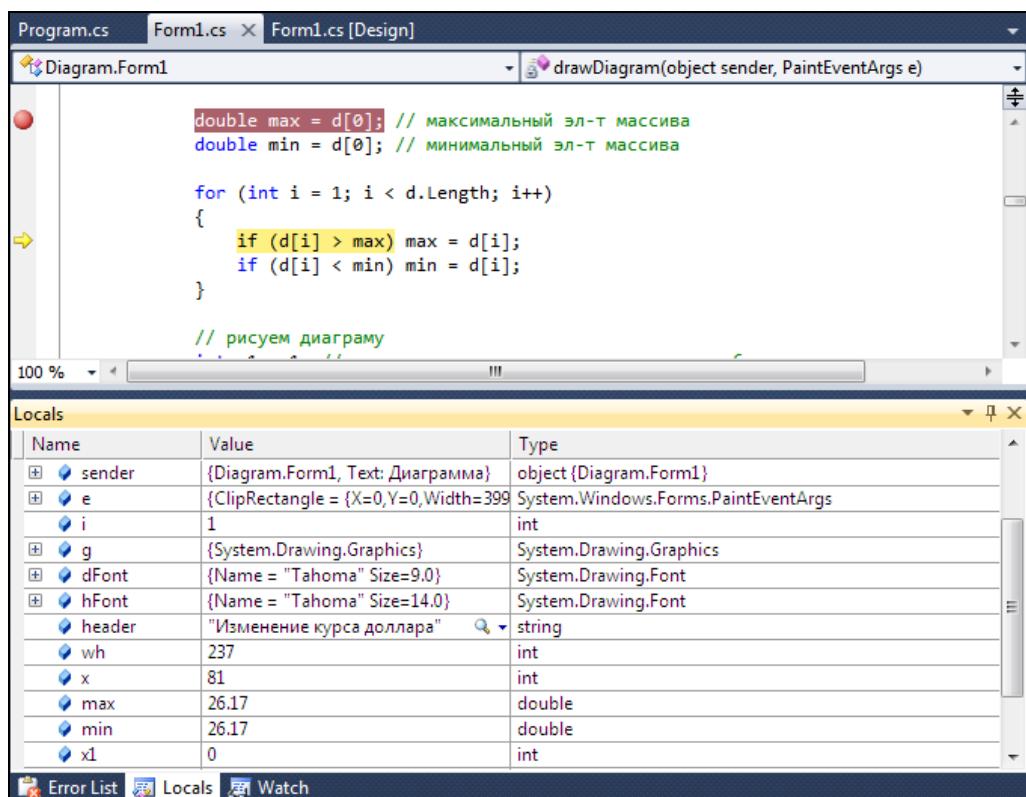


Рис. 8.5. Процесс отладки программы

Если переменных в программе много, то наблюдать значение нужной переменной в окне **Locals** неудобно. Поэтому если необходимо контролировать значения конкретных переменных, то в этом случае программист может создать список наблюдаемых переменных (**Watch**) и во время выполнения программы следить за переменными списка. Чтобы добавить переменную в список наблюдаемых значений, надо на вкладке **Locals** сделать щелчок правой кнопкой мыши на имени нужной переменной и в появившемся контекстном меню выбрать команду **Add Watch**. В результате выбранная переменная будет добавлена в список наблюдаемых значений (рис. 8.6).

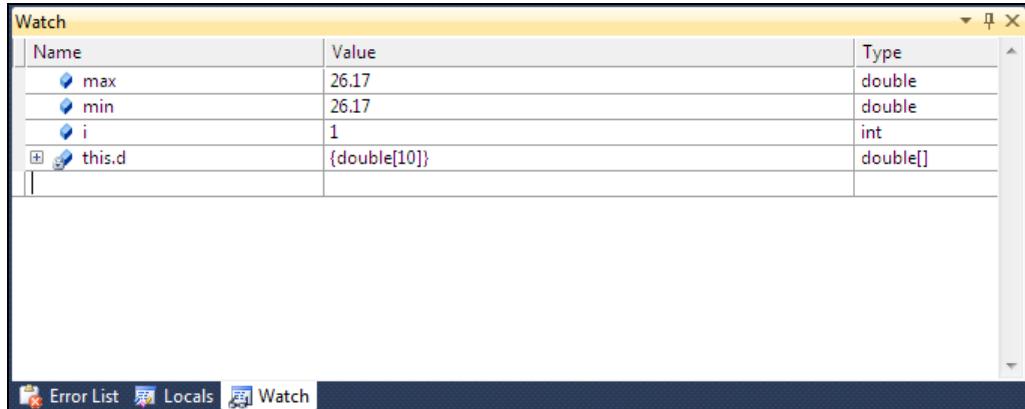


Рис. 8.6. Пример списка наблюдаемых переменных

Другой способ увидеть значение переменной — установить указатель мыши на ее имени (рис. 8.7).

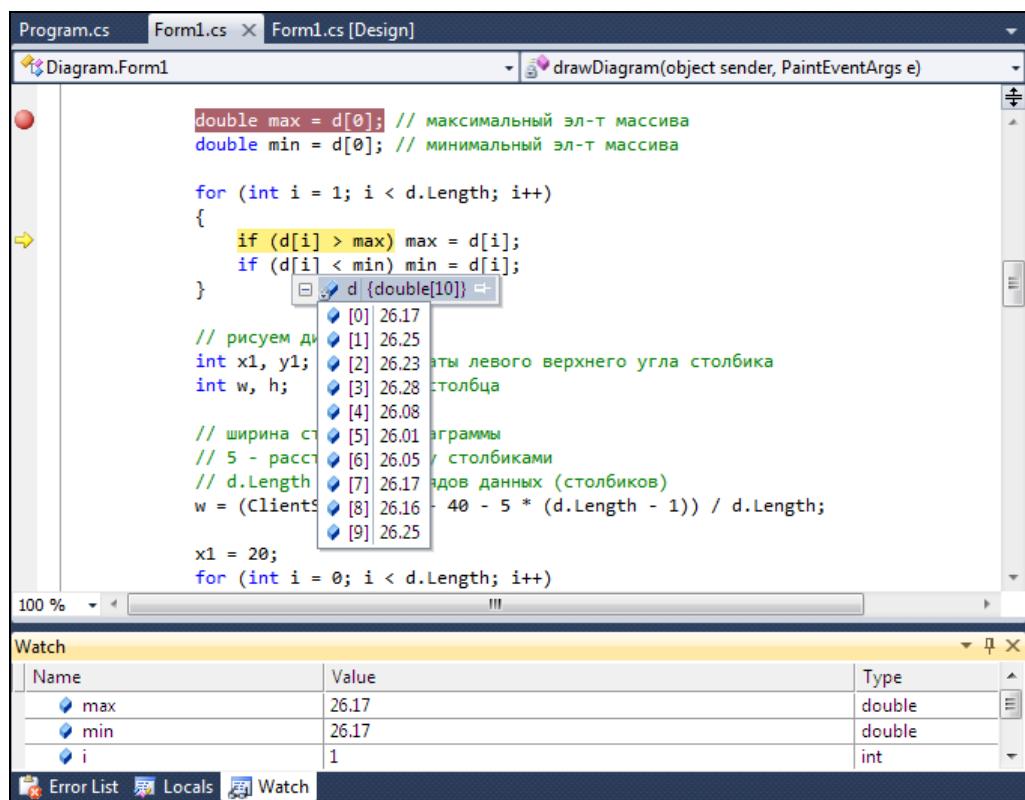
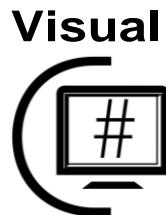


Рис. 8.7. При позиционировании указателя мыши на переменной отображается ее значение



ГЛАВА 9

Справочная информация

Разработчик программы должен позаботиться о том, чтобы пользователю было удобно работать с программой, чтобы при возникновении какой-либо проблемы он мог быстро получить рекомендацию по ее устраниению, ответ на возникший вопрос. Решить обозначенную задачу можно, создав справочную систему, обеспечивающую доступ к справочной информации.

Справочная система HTML Help

В настоящее время в большинстве *прикладных* программ справочная информация отображается в формате HTML Help. По запросу пользователя, в результате нажатия клавиши <F1>, выбора соответствующей команды в меню **Справка** или нажатия кнопки **Справка** в диалоговом окне, на экране появляется окно справочной информации. Оно, как правило, разделено на две части: в правой части отображается список разделов справочной информации (оглавление), в левой — содержание выбранного раздела (рис. 9.1).

Основой справочной системы формата HTML Help являются *компилированные* HTML-документы (chm-файлы) — совокупность отдельных HTML-страниц справочной информации. Помимо HTML-страниц chm-файл обычно содержит список названий разделов справочной информации (оглавление) и предметный указатель (индекс).

Отображение справочной информации формата HTML Help обеспечивает операционная система (утилита hh.exe). Таким образом, в простейшем случае задача создания справочной системы сводится к созданию chm-файла.

Создать chm-файл можно с помощью Microsoft HTML Help Workshop. Сначала надо подготовить справочную информацию (информацию каждого раздела следует поместить в отдельный HTML-файл), затем — выполнить компиляцию.

Процесс преобразования исходной справочной информации в chm-файл представлен на рис. 9.2. Исходной информацией для компилятора являются файл проекта, файлы справочной информации и файл контента. Файлы проекта и контента создаются непосредственно в Microsoft HTML Help Workshop.

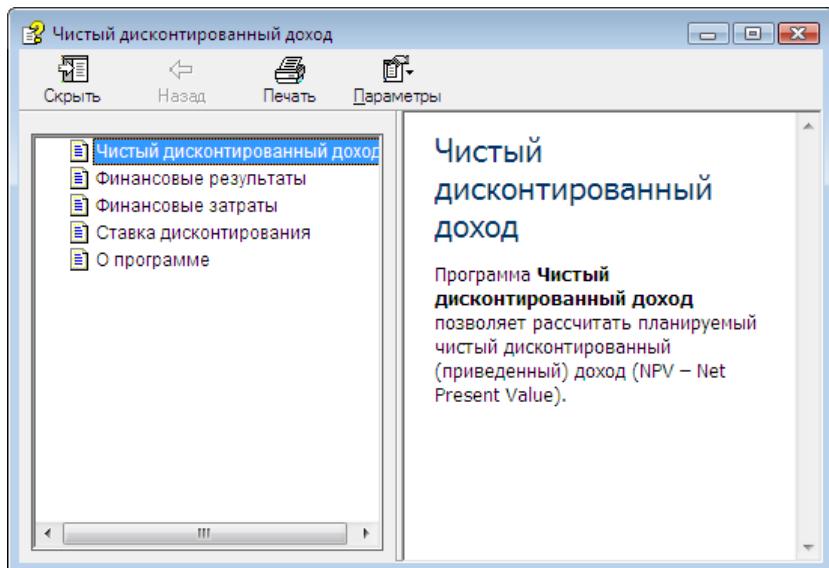


Рис. 9.1. Пример справочной информации в формате HTML Help

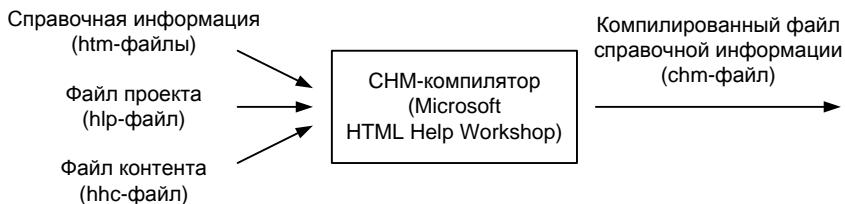


Рис. 9.2. Процесс создания chm-файла

Подготовка справочной информации

Исходным "материалом" для CHM-компилятора является справочная информация, представленная в виде набора HTML-файлов.

Создать файл справочной информации в HTML-формате можно с помощью HTML-редактора, редактора "чистого" текста (например, Блокнота) или Microsoft Word, сохранив набранный текст в формате Web-страницы.

В простейшем случае всю справочную информацию можно поместить в один HTML-файл. Однако если для навигации по справочной системе предполагается использовать вкладку **Содержание**, на которой будут перечислены разделы справочной информации, то информацию каждого раздела нужно поместить в отдельный HTML-файл. Имена файлам рекомендуется дать так, чтобы в них присутствовал номер раздела. Например: npv_01.htm, npv_02.htm и т. д.

Использование для подготовки справочной информации HTML-редактора предполагает знание основ HTML — языка гипертекстовой разметки. Далее приведено

краткое описание HTML в объеме, необходимом для подготовки текста раздела справочной информации.

Основы HTML

HTML-документ представляет собой *текст*, в который помимо обычного текста, предназначенного для восприятия читателем, включены специальные, не отображаемые последовательности символов — *теги*. Теги начинаются символом < и заканчиваются символом >. Они используются программами отображения HTML-документов для форматирования текста в окне просмотра.

Большинство тегов парные. Например, пара тегов <h1> </h1> сообщает программе отображения HTML-документа, что текст, находящийся между ними, является заголовком первого уровня и должен быть отображен соответствующим стилем.

В табл. 9.1 представлен минимальный набор тегов, используя которые можно подготовить HTML-файл справочной информации.

Таблица 9.1. HTML-теги

Тег	Пояснение
<TITLE> Название </TITLE>	Задает название HTML-документа. Программы отображения HTML-документов, как правило, выводят название документа в заголовке окна, в котором документ отображается. Если название не задано, то в заголовке окна будет выведено название файла
<BASEFONT FACE="Шрифт" SIZE=n>	Задает основной шрифт, который используется для отображения текста: FACE — название шрифта, SIZE — размер в относительных единицах. По умолчанию значение параметра SIZE равно 3. Размер шрифта заголовков (см. тег <h>) берется от размера, заданного параметром SIZE
<H1> </H1>	Определяет текст, находящийся между тегами <h1> и </h1> как заголовок уровня 1. Пара тегов <h2></h2> определяет заголовок второго уровня, а пара <h3></h3> — третьего
 	Конец строки. Текст, находящийся после этого тега, будет выведен с начала новой строки
<P> </P>	Текст, находящийся внутри этих тегов, является параграфом
 	Текст, находящийся внутри этой пары тегов, будет выделен полужирным
<I> </I>	Текст, находящийся внутри этой пары тегов, будет выделен курсивом
 	Закладка. Помечает точку документа закладкой. Имя закладки задает параметр NAME. Это имя используется для перехода к закладке (фрагменту документа) в результате выбора ссылки

Таблица 9.1 (окончание)

Тег	Пояснение
 	Выделяет фрагмент документа как гиперссылку, при выборе которой происходит перемещение к закладке, имя которой указано в параметре <code>HREF</code>
	Выводит иллюстрацию, имя файла которой указано в качестве значения параметра <code>SRC</code>
<!-- -->	Комментарий. Текст, находящийся между дефисами, на экран не выводится

Набирается HTML-текст обычным образом. Теги можно набирать как прописными, так и строчными буквами. Однако, чтобы лучше была видна структура документа, рекомендуется записывать все теги прописными (большими) буквами. Следующее, на что надо обратить внимание, — программы отображения HTML-документов игнорируют "лишние" пробелы и другие "невидимые" символы (табуляцию, новые строки). Это значит, что для того чтобы фрагмент документа начался с новой строки, в конце предыдущей строки надо поставить тег `
`, а чтобы между строками текста появилась пустая строка, в HTML-текст надо вставить два тега `
` подряд.

В качестве примера в листинге 9.1 приведен HTML-текст одного из разделов справочной системы программы "Чистый дисконтированный доход".

Листинг 9.1. Раздел справочной информации (prv_04.htm)

```

<HTML>
<HEAD>
<TITLE>Ставка дисконтирования</TITLE>
</HEAD>
<BODY><A NAME="Ставка_дисконтирования"></A>
<H2>Ставка дисконтирования</H2>
<P>Ставка дисконтирования позволяет учесть изменение стоимости денег
во времени, привести будущие денежные потоки к настоящему моменту. В первом
приближении в качестве значения ставки дисконтирования можно принять значение
показателя инфляции.
</P>
<P>Введите значение ставки дисконтирования в процентах (знак процента вводить
не надо).
</P>
</BODY>
</HTML>

```

Microsoft HTML Help Workshop

Как было сказано, создать справочную систему формата HTML Help можно с помощью Microsoft HTML Help Workshop.

Чтобы создать справочную систему, надо:

1. Создать файл проекта.
2. Создать файл контента.
3. Создать файл индекса.
4. Выполнить компиляцию.

Файл проекта

Чтобы начать работу по созданию справочной системы, надо запустить Microsoft HTML Help Workshop, в меню **File** выбрать команду **New ▶ Project**, затем в окне **New Project** задать имя файла проекта (файл проекта надо создать в том каталоге, в котором находятся HTML-файлы страниц справочной информации). Следует обратить внимание, что по умолчанию имя chm-файла, который будет создан в процессе компиляции, совпадает с именем файла проекта.

В качестве примера на рис. 9.3 приведено окно **HTML Help Workshop** в начале работы над проектом справочной системы программы "Чистый дисконтированный доход".

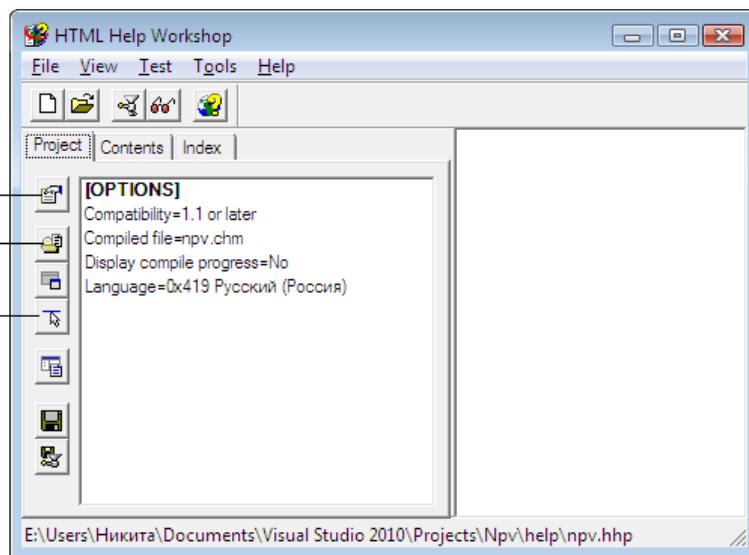


Рис. 9.3. Окно **HTML Help Workshop** в начале работы над новым проектом

После того как файл проекта будет создан, надо сформировать список файлов, в которых находится справочная информация (предполагается, что все необходимые HTML-файлы находятся в одном каталоге, в том же, в котором сохранен файл проекта).

Чтобы указать файлы, в которых находится справочная информация, надо:

1. Щелкнуть на кнопке **Add topic files**.
2. В появившемся окне **Topic Files** щелкнуть на кнопке **Add**.

3. В стандартном окне **Открыть** выбрать HTML-файлы, в которых находится справочная информация (нажать клавишу <Ctrl> и, удерживая ее нажатой, щелкнуть на именах нужных файлов).

В результате описанных действий в окне **Topic Files** появится список файлов (рис. 9.4), в которых находится справочная информация, а после щелчка на кнопке **OK** в файл проекта (его содержимое отображается на вкладке **Project**) будет добавлен раздел **FILES**, в котором будут перечислены HTML-файлы, содержащие справочную информацию (рис. 9.5).

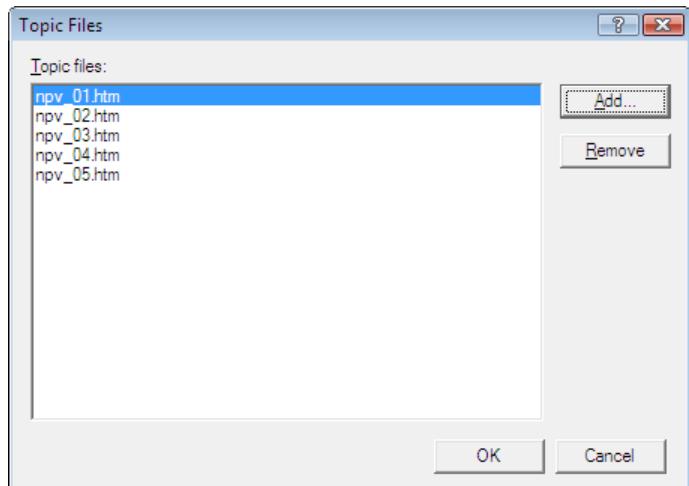


Рис. 9.4. Формирование списка файлов, в которых находится справочная информация

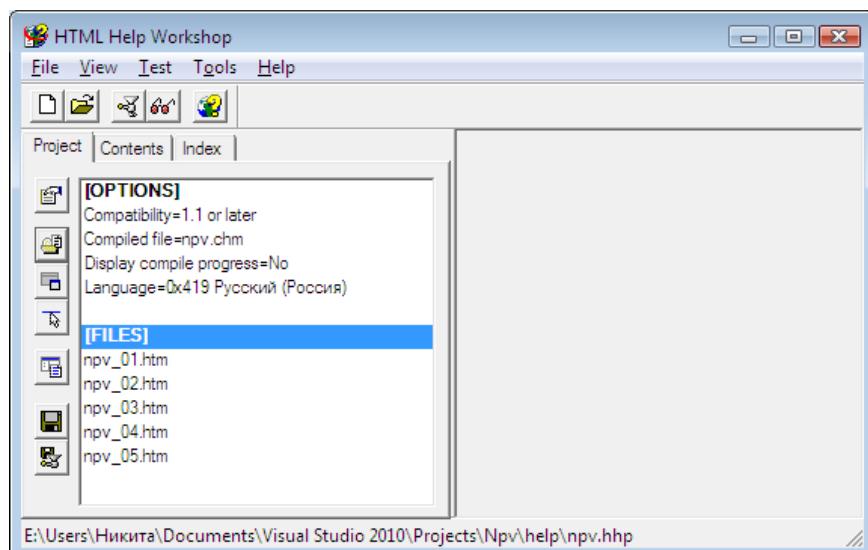


Рис. 9.5. В разделе FILES перечислены файлы, в которых находится справочная информация

Следующее, что нужно сделать, — задать главную (стартовую) страницу и заголовок окна справочной информации. Заголовок окна и имя файла главной страницы надо ввести соответственно в поля **Title** и **Default file** вкладки **General** окна **Options** (рис. 9.6), которое становится доступным в результате щелчка на кнопке **Change project options** (см. рис. 9.3).

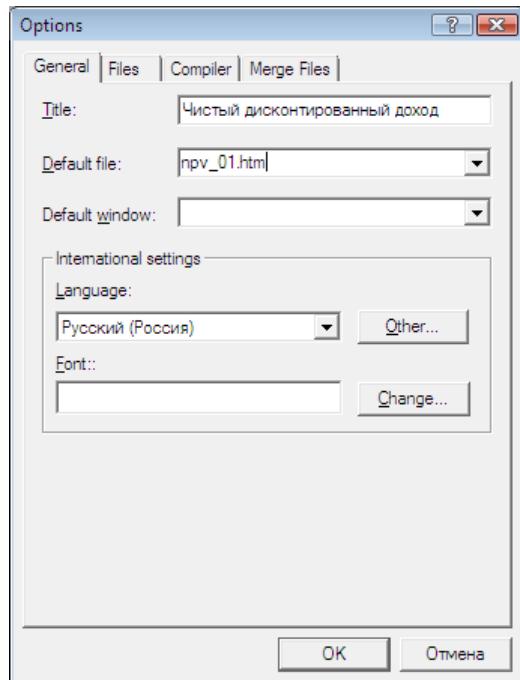


Рис. 9.6. В поле **Title** надо ввести заголовок окна справочной информации, а в поле **Default file** — имя файла главной страницы

Оглавление

Если для навигации по справочной системе предполагается использовать вкладку **Оглавление**, то надо создать *файл контента*. Чтобы это сделать, нужно щелкнуть на ярлыке вкладки **Contents**, подтвердить создание нового файла контента (**Create a new contents file**) и задать его имя (в качестве имени файла контента рекомендуется указать имя проекта). В результате выполнения описанных действий в каталоге проекта будет создан файл контента (hhc-файл) и станет доступной вкладка **Contents** (рис. 9.7).

Оглавление справочной информации формируется путем добавления на вкладку **Contents** ссылок на страницы справочной информации.

Чтобы на вкладку **Contents** добавить ссылку на страницу справочной информации, надо:

1. Щелкнуть на кнопке **Insert a page**.
2. В поле **Entry title** вкладки **General** открывшегося окна **Table of Contents Entry** ввести название раздела справочной информации и щелкнуть на кнопке **Add**.

(рис. 9.8). В результате станет доступным окно **Path or URL** (рис. 9.9), в списке **HTML titles** которого будут перечислены названия HTML-документов, включенных в проект (если в HTML-файле нет тега <Title>, то вместо названия документа отображается имя файла).

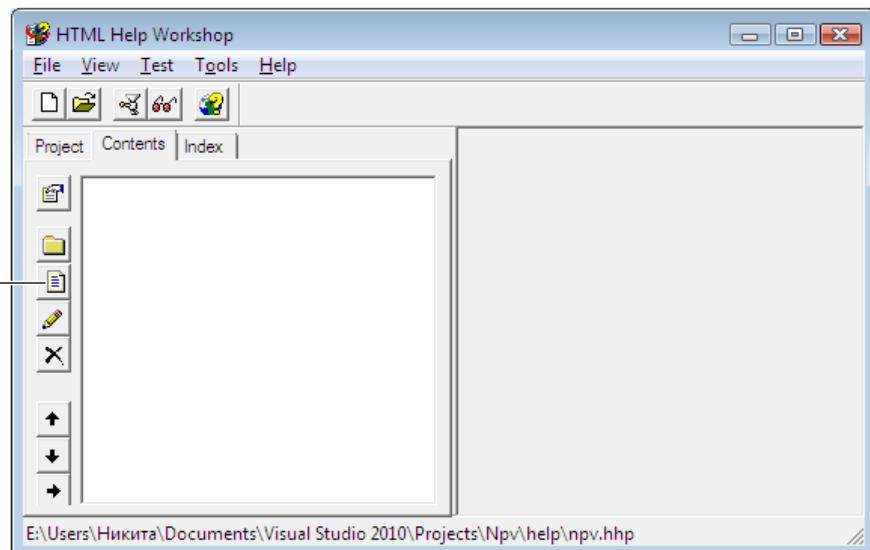


Рис. 9.7. Вкладка Contents

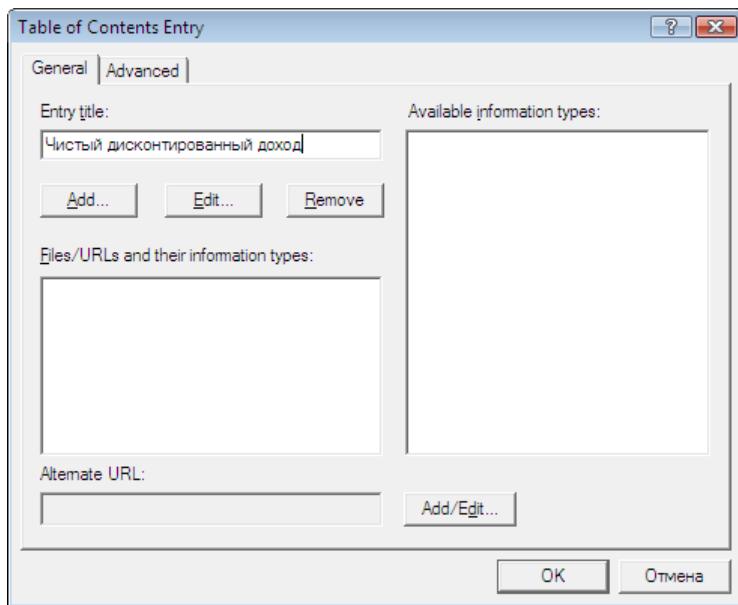


Рис. 9.8. В поле **Entry title** надо ввести название раздела и щелкнуть на кнопке **Add**

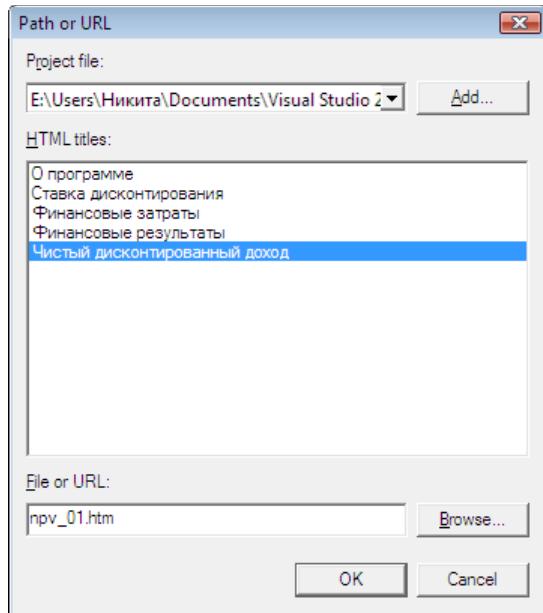


Рис. 9.9. В списке **HTML titles** надо выбрать HTML-страницу

3. В списке **HTML titles** окна **Path or URL** выбрать HTML-документ, который надо отобразить в окне справочной информации в результате выбора ссылки на страницу (названия раздела справочной информации).
4. Щелкнуть на кнопке **OK**.

Указанную последовательность действий надо выполнить для каждого раздела справочной информации.

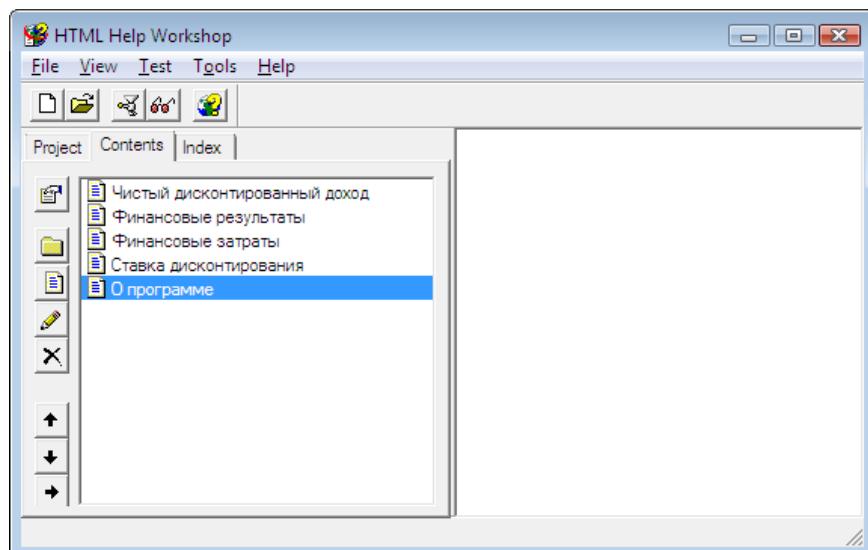


Рис. 9.10. Вкладка **Contents** содержит оглавление справочной информации

В качестве примера на рис. 9.10 приведено окно **HTML Help Workshop**, вкладка **Contents** которого содержит оглавление справочной информации программы "Чистый дисконтированный доход".

Идентификаторы разделов

Обычно когда пользователь, нажав клавишу <F1>, запрашивает справочную информацию, в окне справочной системы сразу отображается нужный раздел (страница). Такой режим отображения справочной информации называется контекстно-зависимым. Для того чтобы программист мог реализовать контекстно-зависимый режим отображения справочной информации, необходимо каждой странице назначить идентификатор.

Инструкция назначения идентификатора в общем виде выглядит так:

```
#define Страница Идентификатор; комментарий
```

Например, инструкция

```
#define npv_01 1; Чистый дисконтированный доход
```

объявляет идентификатор для страницы npv_01.htm. Следует обратить внимание, что при объявлении идентификатора расширение файла не указывается.

Инструкции объявления идентификаторов следует поместить в отдельный файл. В качестве примера в листинге 9.2 показан файл объявления идентификаторов разделов справочной информации программы "Чистый дисконтированный доход". Следует обратить внимание, что файл с расширением h — это обычный текстовый файл.

Листинг 9.2. Объявление идентификаторов разделов справочной информации (profit.h)

```
#define npv_01 1; Чистый дисконтированный доход
#define npv_02 2; Финансовые результаты
#define npv_03 3; Финансовые затраты
#define npv_04 4; Ставка дисконтирования
#define npv_05 5; О программе
```

После того как h-файл будет подготовлен (его следует сохранить в каталоге проекта справочной системы), надо:

1. Выбрать вкладку **Project** и сделать щелчок на кнопке **HTML Help API Information**.
2. Выбрать вкладку **Map** и щелкнуть на кнопке **Header file**.
3. В появившемся окне **Include file** ввести имя h-файла и щелкнуть на кнопке **OK**.

В результате описанных действий в файл проекта будет добавлен раздел **MAP** (рис. 9.11), в котором будет ссылка (директива `#include`) на файл объявления идентификаторов разделов справочной информации.

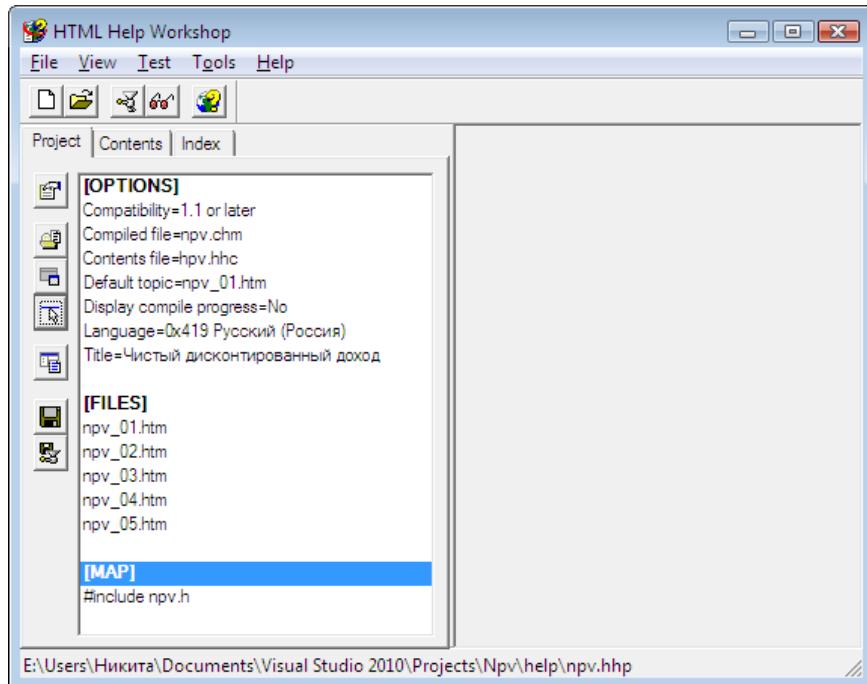


Рис. 9.11. Раздел MAP содержит ссылку на файл идентификаторов разделов справочной информации

Компиляция

После того как будут определены файлы, в которых находится справочная информация, и подготовлена информация для формирования вкладки **Содержание** (создан файл контента), можно выполнить компиляцию.

Исходной информацией для CHM-компилятора HTML Help Workshop являются файл проекта (hpp-файл), файл контента (hhc-файл) и файлы справочной информации (html-файлы). Результатом компиляции является файл справочной информации (chm-файл).

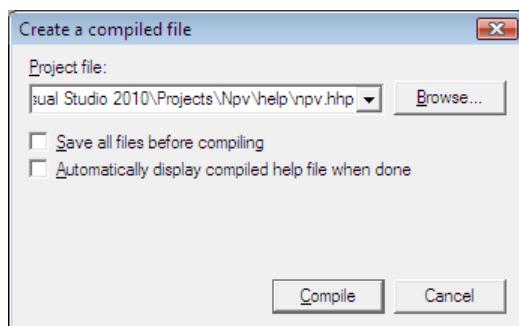


Рис. 9.12. Чтобы создать chm-файл, надо щелкнуть на кнопке Compile

Чтобы выполнить компиляцию, надо в меню **File** выбрать команду **Compile** и затем в появившемся окне **Create a compiled file** сделать щелчок на кнопке **Compile** (рис. 9.12).

В результате в каталоге проекта будет создан chm-файл справочной информации. Чтобы убедиться, что файл справочной информации создан правильно, надо в меню **View** выбрать команду **Compiled Help File**. На экране должно появиться окно справочной информации.

Доступ к файлу справочной информации

Для того чтобы файл справочной информации был гарантированно доступен приложению, его следует поместить в тот каталог, в котором находится выполняемый файл. При этом следует обратить внимание на то, что в зависимости от текущего режима компиляции (Debug или Release) компилятор помещает выполняемый файл в папку Debug или Release. Можно, конечно, вручную скопировать файл справки в оба каталога. Однако лучше поступить иначе — сделать так, чтобы файл справки автоматически копировался в нужный каталог. Чтобы это происходило, сначала надо в проект добавить файл справочной информации — в меню **Project** выбрать команду **Add Existing Item** и в открывшемся окне указать файл справки. Затем следует в окне **Solution Explorer** (рис. 9.13) выбрать добавленный файл и в окне **Properties** в качестве значения свойства **Copy to Output Directory** выбрать **Copy if newer** (рис. 9.14). Кроме того, чтобы файл справки был включен в дистрибутив для публикации приложения (см. главу 10) свойству **Build Action** файла справки следует присвоить значение **Content** (рис. 9.15).

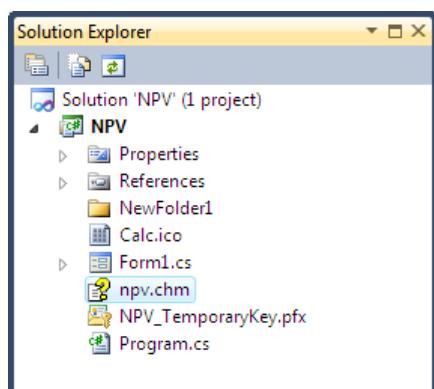


Рис. 9.13. Имя добавленного в проект файла справочной информации отображается в окне **Solution Explorer**

Отображение справочной информации

Отображение справочной информации обеспечивает метод `ShowHelp()` объекта `Help`, которому в качестве параметра передается идентификатор объекта (формы), запрашивающего справочную информацию, имя chm-файла справочной информа-

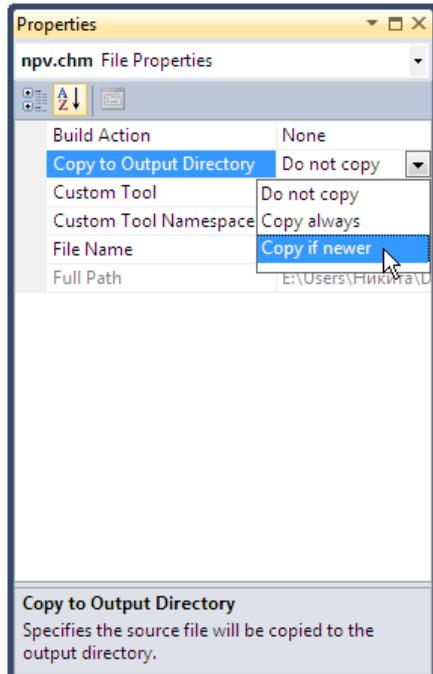


Рис. 9.14. Чтобы файл справки был автоматически помещен в папку Debug или Release, выберите **Copy if newer**

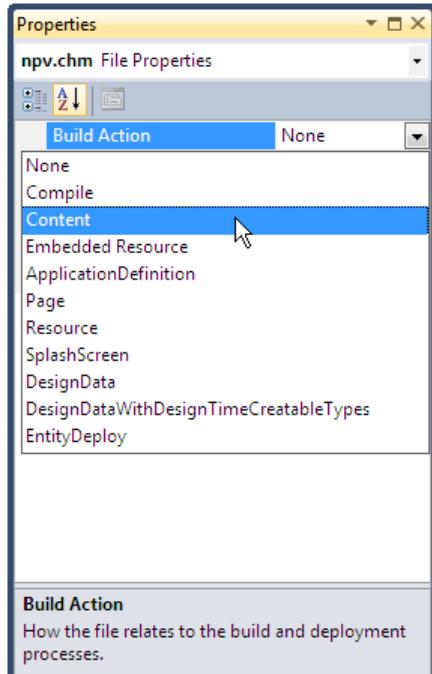


Рис. 9.15. Чтобы файл справки был помещен в дистрибутив, свойству **Build Action** надо присвоить значение **Content**

ции (если файл находится в той же папке, что и выполняемый файл программы, путь к файлу можно не указывать) и идентификатор раздела справочной информации. В качестве примера в листинге 9.3 приведена процедура обработки щелчка (события Click) на кнопке **Справка** окна программы "Чистый дисконтированный доход".

Листинг 9.3. Отображение справочной информации

```
private void button2_Click(object sender, EventArgs e)
{
    Help.ShowHelp(this, "npv.chm", "npv_01.htm");
}
```

Общепринятым способом доступа к справочной информации является нажатие клавиши <F1>. Причем, как правило, при нажатии <F1> в окне справочной информации отображается раздел справки, связанный действиями пользователя, которые выполняются в данный момент (такой способ доступа к справочной информации называют *контекстно-зависимым*).

Для того чтобы реализовать этот способ отображения справочной информации, на форму надо добавить компонент HelpProvider (рис. 9.16).

После того как компонент HelpProvider будет добавлен на форму, надо выполнить его настройку — указать имя файла справочной информации в качестве значения свойства HelpNamespace. Далее для каждого элемента управления надо указать раздел справочной информации, который следует отобразить в результате нажатия клавиши <F1> в момент нахождения фокуса на этом элементе. Раздел указывается путем установки значений свойств ShowHelp, HelpNavigator и HelpKeyWord.

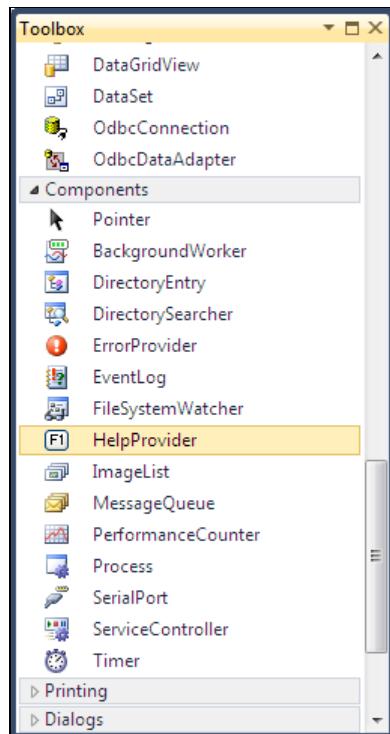


Рис. 9.16. Компонент HelpProvider

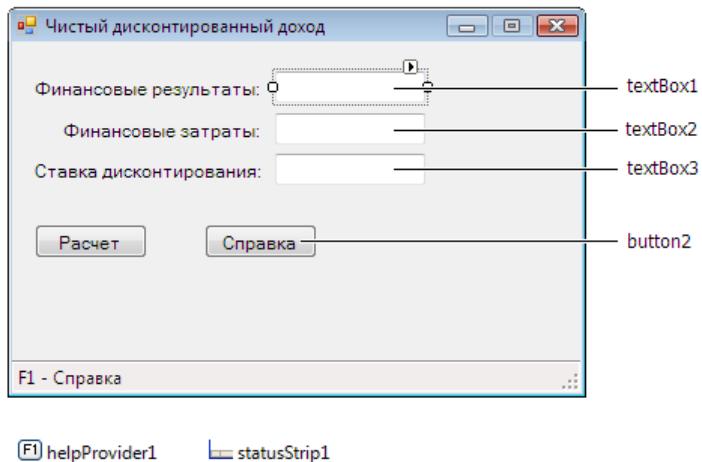


Рис. 9.17. Справочная информация отображается в результате нажатия клавиши <F1> или щелчка на кнопке Справка

Отображение справочной информации демонстрирует программа "Чистый дисконтированный доход" (ее форма приведена на рис. 9.17). Во время работы с программой пользователь может получить справку, сделав щелчок на кнопке **Справка** или нажав клавишу <F1>. В первом случае отображается главный раздел справочной информации, во втором, в зависимости от того, в какое поле в данный момент

пользователь вводит информацию, — раздел **Финансовые результаты**, **Финансовые затраты** или **Ставка дисконтирования**. Значения свойств компонентов приведены в табл. 9.2.

Таблица 9.2. Значения свойств компонентов

Компонент	Свойство	Значение
helpProvider1	HelpNamespace	npv.chm
textBox1	ShowHelp on helpProvider1	True
	HelpNavigator on helpProvider1	Topic
	HelpKeyWord on helpProvider1	npv_02.htm
textBox2	ShowHelp on helpProvider1	True
	HelpNavigator on helpProvider1	Topic
	HelpKeyWord on helpProvider1	npv_03.htm
textBox2	ShowHelp on helpProvider1	True
	HelpNavigator on helpProvider1	Topic
	HelpKeyWord on helpProvider1	npv_04.htm

В листинге 9.4 приведен конструктор формы и функции обработки события Click на кнопках **Расчет** и **Справка** программы "Чистый дисконтированный доход". Конструктор показывает, как можно задать значения свойств, определяющих отображаемый раздел справочной информации, в коде.

**Листинг 9.4. Чистый дисконтированный доход
(отображение справочной информации)**

```
public Form1()
{
    InitializeComponent();

    // Файл справки
    helpProvider1.HelpNamespace = "npv.chm";
    helpProvider1.SetHelpNavigator(this, HelpNavigator.Topic);
    helpProvider1.SetShowHelp(this, true);

    // Chm-файл получается путем компиляции htm-файлов,
    // в которых находится справочная информация.
    // Обычно каждый раздел справочной информации помещают в отдельный файл.
    // В дальнейшем имя htm-файла используется в качестве идентификатора
    // раздела справочной информации.

    // Задать раздел справки
    // для textBox1 - Финансовые результаты
    helpProvider1.SetHelpKeyword(textBox1, "npv_02.htm");
```

```
helpProvider1.SetHelpNavigator(textBox1, HelpNavigator.Topic);
helpProvider1.SetShowHelp(textBox1, true);

// Задать раздел справки
// для textBox2 - Финансовые затраты
helpProvider1.SetHelpKeyword(textBox2, "npv_03.htm");
helpProvider1.SetHelpNavigator(textBox2, HelpNavigator.Topic);
helpProvider1.SetShowHelp(textBox2, true);

// Задать раздел справки
// для textBox3 - Ставка дисконтирования
helpProvider1.SetHelpKeyword(textBox3, "npv_04.htm");
helpProvider1.SetHelpNavigator(textBox3, HelpNavigator.Topic);
helpProvider1.SetShowHelp(textBox3, true);

}

// Щелчок на кнопке Расчет
private void button1_Click(object sender, EventArgs e)
{
    double p = 0; // поступления от продаж
    double r = 0; // расходы
    double d = 0; // ставка дисконтирования

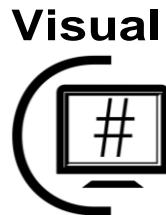
    double npv = 0; // чистый дисконтированный доход

    p = Convert.ToDouble(textBox1.Text);
    r = Convert.ToDouble(textBox2.Text);
    d = Convert.ToDouble(textBox3.Text) / 100;

    npv = (p - r) / (1.0 + d);

    label4.Text = "Чистый дисконтированный доход (NPV) = " +
        npv.ToString("c");
}

// Щелчок на кнопке Справка
private void button2_Click(object sender, EventArgs e)
{
    Help.ShowHelp(this, helpProvider1.HelpNamespace, "npv_01.htm");
}
```



ГЛАВА 10

Публикация приложения

Готовую, отлаженную программу можно передать всем, кому она нужна. Сделать это можно, например, с помощью флэш-накопителя (флэшки), скопировав на него файлы подкатаога Release каталога проекта. Пользователь должен будет переместить файлы с флэшки на жесткий диск своего компьютера. Эта операция, очевидно, с легкостью будет выполнена программистом, но у обычного пользователя могут возникнуть трудности. Поэтому лучше следовать принятой практике, когда установку прикладной программы на компьютер пользователя выполняет специальная программа — установщик, который не только копирует необходимые файлы с промежуточного носителя на жесткий диск компьютера пользователя, но и, например, создает на рабочем столе ярлык, обеспечивающий запуск установленной программы.

Для установки приложений на компьютер пользователя Microsoft рекомендует использовать одну из двух технологий: Windows Installer или ClickOnce.

Технология Windows Installer предполагает создание базы данных специального формата (msi-файла). В этой базе данных находятся файлы, которые нужно перенести на компьютер пользователя, и информация о необходимых настройках. Установку приложения выполняет Windows Installer путем обработки msi-файла.

Технология ClickOnce (что можно перевести, как "один (или единственный) щелчок") была разработана специально для развертывания .NET-приложений. Установщик формата ClickOnce можно создать при помощи мастера публикации (Publish Wizard), который доступен из Visual C#.

Подготовка к публикации

Процесс подготовки приложения для передачи пользователю, точнее, создания программы-установщика, называется *публикацией*.

Microsoft Visual C# 2010 позволяет подготовить приложение для размещения на CD или на Web-сайте.

Процесс подготовки приложения для публикации (размещения) на CD рассмотрим на примере программы NPV.

Список файлов, которые необходимо перенести на компьютер пользователя, приведен в табл. 10.1.

Таблица 10.1. Файлы, которые нужно установить на компьютер пользователя

Файл	Описание
prv.exe	Выполняемый файл приложения (программа)
prv.chm	Файл справочной информации

По умолчанию приложение Visual C# компилируется в режиме Debug (Отладка), поэтому, перед тем как приступить непосредственно к публикации приложения, необходимо выполнить его компиляцию в режиме Release (Выпуск).

Режим компиляции задается на вкладке **Build** окна **Properties**, которое становится доступным в результате выполнения команды **Project ▶ Properties**, путем выбора в раскрывающемся списке **Configuration** соответствующего значения (рис. 10.1).

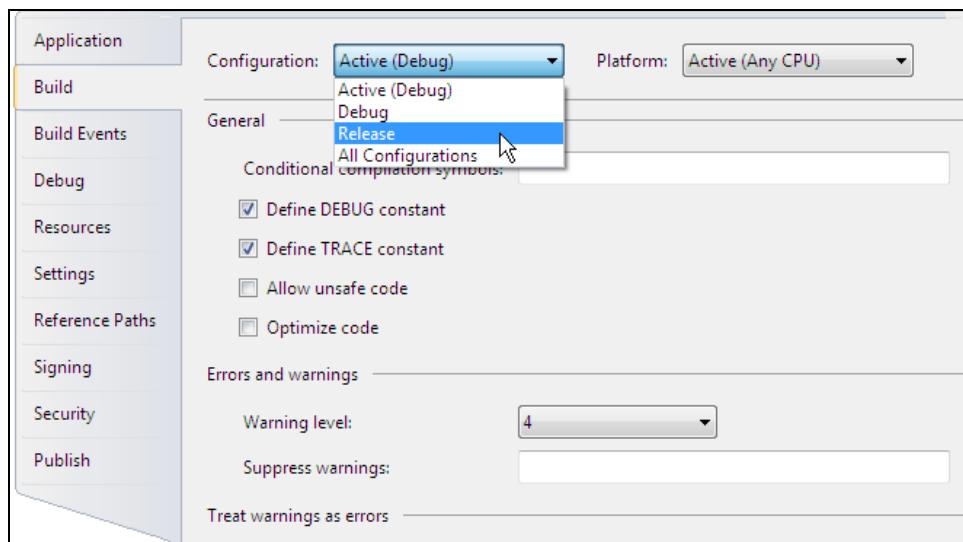


Рис. 10.1. Выбор режима компиляции

Кроме этого рекомендуется на вкладке **Application** задать значок, который будет изображать приложение на рабочем столе или в меню **Программы** компьютера пользователя, а также на вкладке **Publish** указать версию и выбрать **Automatically increment revision with each publish**.

Сделав щелчок на находящейся на вкладке **Publish** кнопке **Application Files**, можно увидеть список публикуемых файлов (рис. 10.2). Чтобы файл был включен в список публикуемых файлов, его надо добавить в проект и свойству **Build Action** присвоить значение **Content**.

Следующее, что надо сделать, — задать имя публикуемого приложения и имя разработчика, а также указать на необходимость создания на рабочем столе ярлыка

запуска. Для этого следует нажать находящуюся на вкладке **Publish** кнопку **Options** и далее в разделе **Description** окна **Publish Options** в поля **Publisher name** и **Product name** ввести соответственно имя разработчика и имя публикуемого приложения (рис. 10.3). А в разделе **Manifests** (рис. 10.4) надо установить флажок **Create desktop shortcut** (Создать ярлык на рабочем столе).

После внесения всех изменений надо выполнить компиляцию программы — выбрать в меню **Debug** команду **Build Solution**.

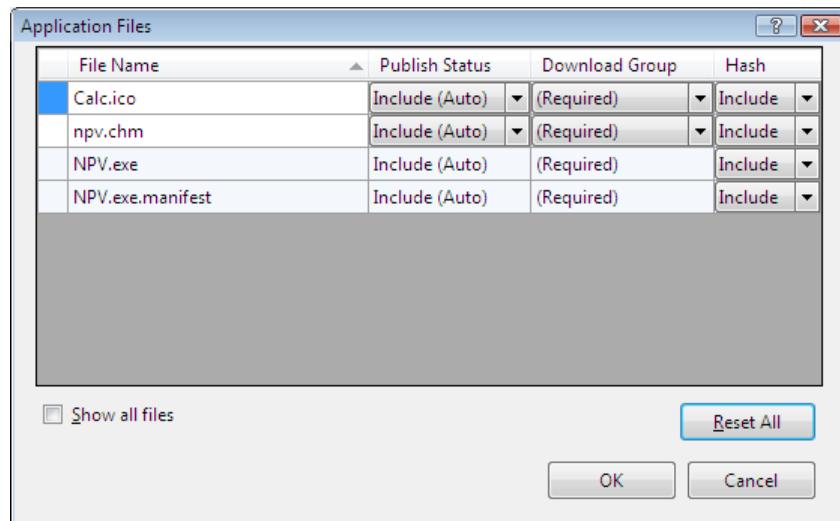


Рис. 10.2. Список публикуемых файлов

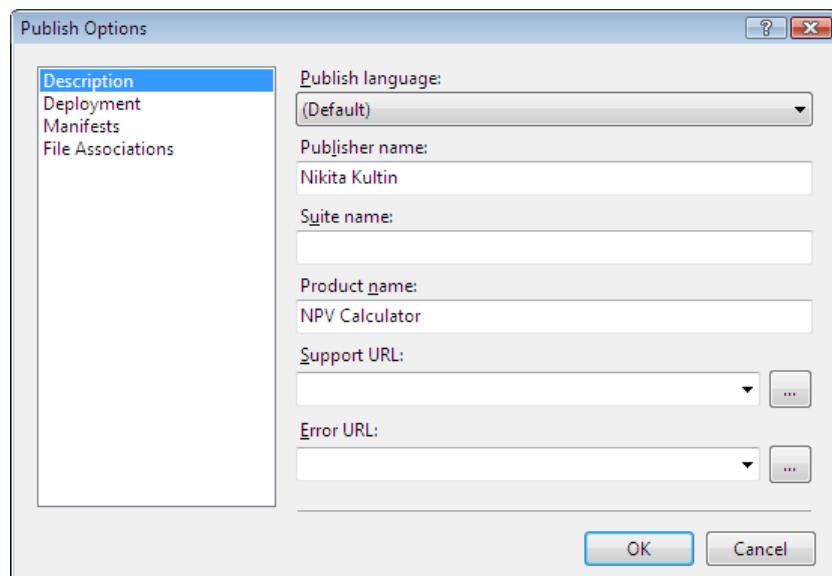


Рис. 10.3. Окно **Publish Options**, раздел **Description**

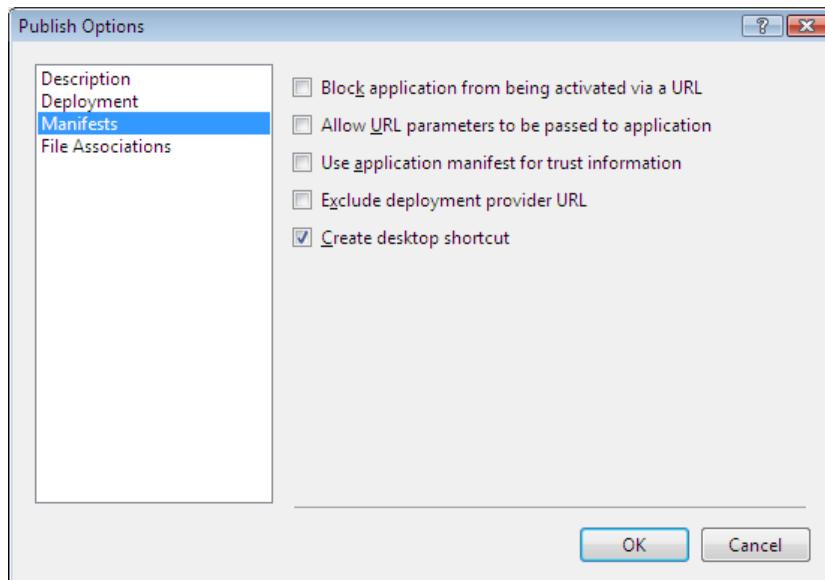


Рис. 10.4. Окно Publish Options, раздел **Manifests**

Мастер публикации

Мастер публикации запускается выбором в меню **Project** команды **Publish**.

В первом окне мастера (рис. 10.5) надо задать каталог, в который будут помещены файлы, предназначенные для публикации. По умолчанию они будут помещены

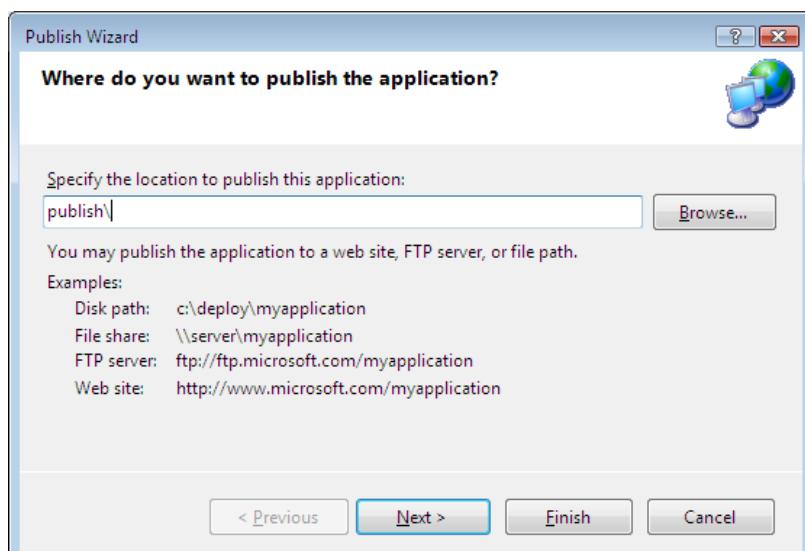


Рис. 10.5. Мастер публикации. Шаг 1

в подкаталог publish каталога проекта. Для перехода к следующему шагу надо нажать кнопку **Next**.

На втором шаге мастера надо выбрать предполагаемое место размещения публикуемых файлов. В данном случае — **From a CD-ROM or DVD-ROM** (рис. 10.6).

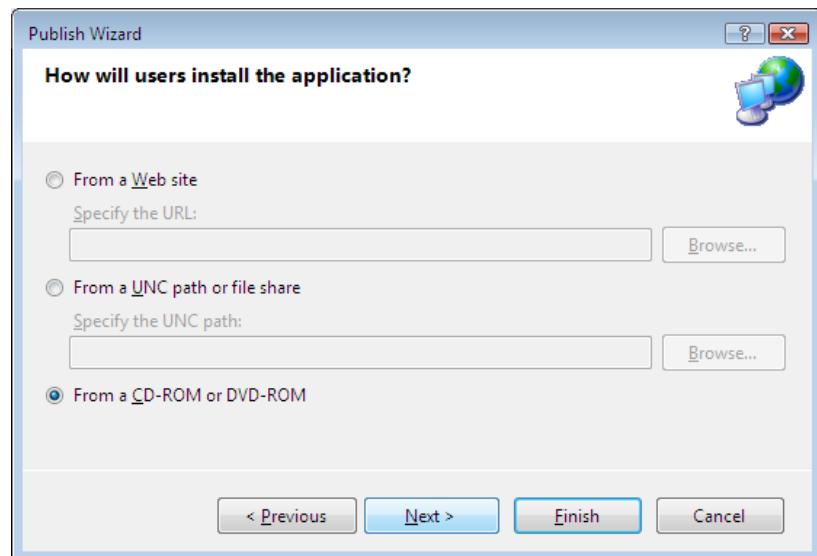


Рис. 10.6. Мастер публикации. Шаг 2

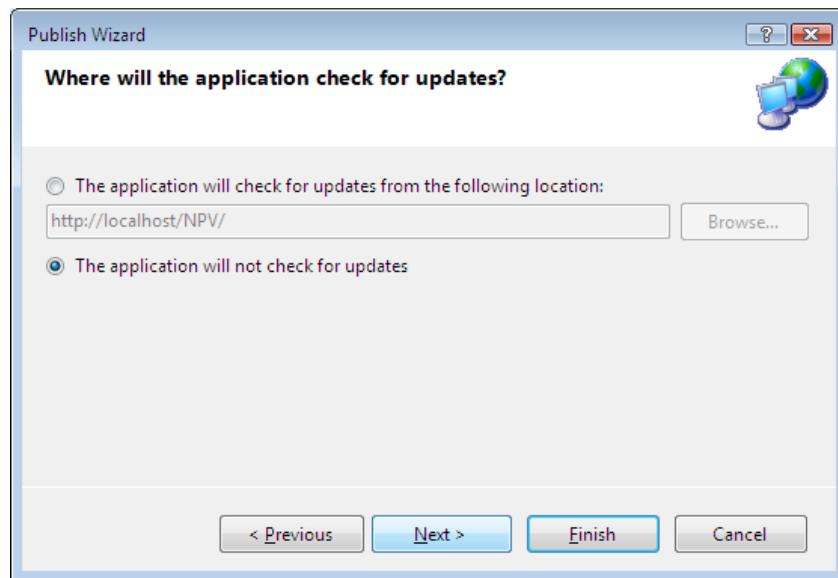


Рис. 10.7. Мастер публикации. Шаг 3

На следующем шаге надо задать, должна ли программа во время своей работы проверять наличие обновлений (рис. 10.7). Если программист не предполагает пре-

доставлять пользователям обновления, то надо выбрать **The application will not check for updates** (Приложение не будет проверять наличие обновлений).

На четвертом шаге (рис. 10.8) надо активизировать процесс публикации — нажать кнопку **Finish**.

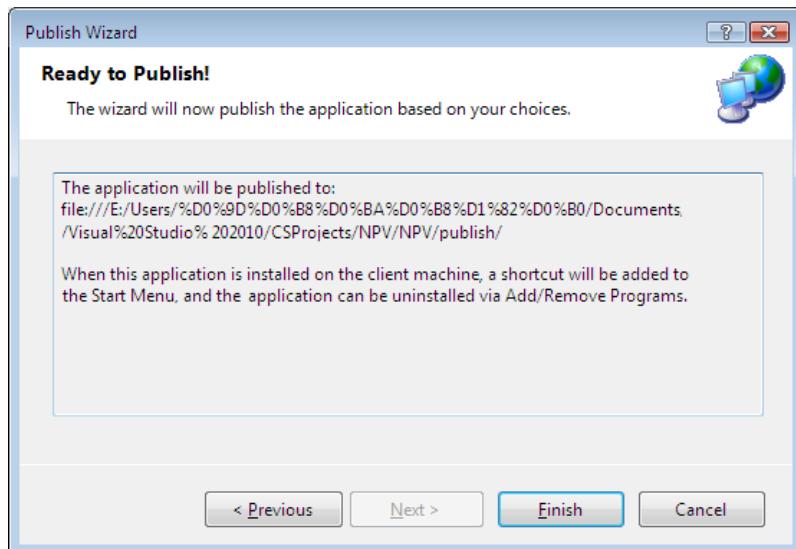


Рис. 10.8. Мастер публикации. Шаг 4

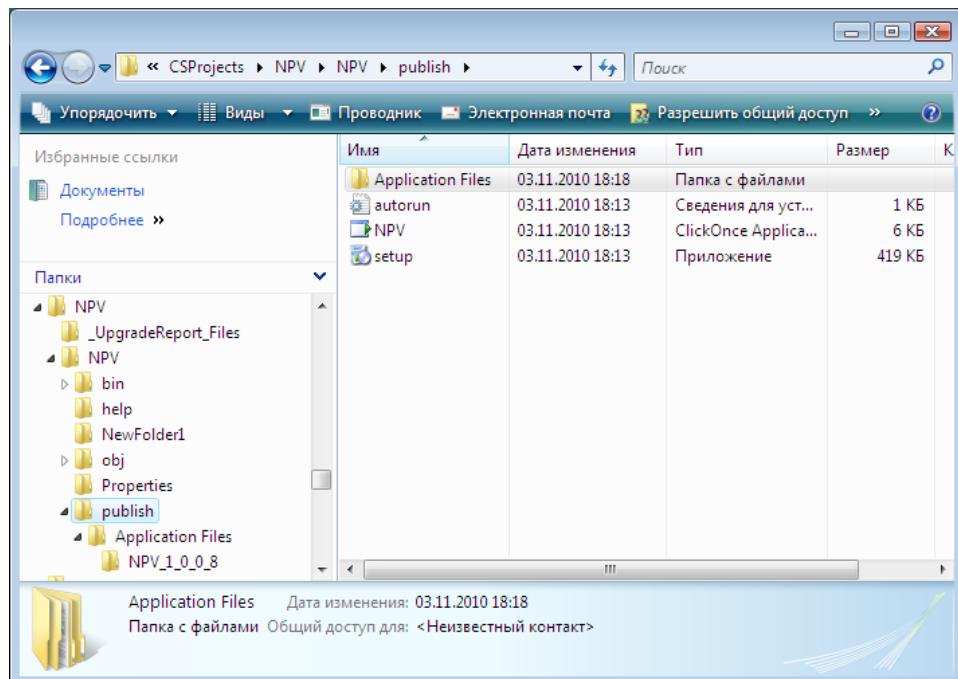


Рис. 10.9. Файлы, предназначенные для публикации

В результате описанных действий в каталог, указанный на шаге 1, будут помещены файлы, предназначенные для публикации (рис. 10.9). Теперь, чтобы передать программу пользователю, достаточно скопировать содержимое каталога publish на CD или флэшку.

Установка

Чтобы активизировать процесс установки, надо запустить программу-установщик (файл setup.exe). Установщик скопирует с CD или флэшки файлы программы на компьютер пользователя, проверит наличие необходимых для работы программы компонентов (в случае отсутствия нужной версии Framework предложит установить ее), создаст на рабочем столе и в меню **Пуск** ярлык, обеспечивающий запуск программы.

По завершении процесса установки программы следует убедиться, что создан ярлык (в папке меню **Пуск** или на рабочем столе), обеспечивающий запуск приложения, щелчком на ярлыке запустить программу и удостовериться, что она работает.

Далее следует убедиться, что установленную программу можно удалить с компьютера. Чтобы это сделать, надо через Панель управления открыть окно **Программы и компоненты**, найти установленную программу в списке программ, сделать двойной щелчок на ее значке, выбрать **Remove application from this computer** и подтвердить удаление.

Если программа установки на компьютере разработчика работает правильно (обеспечивает установку и удаление приложения), то необходимо проверить, как она работает на другом компьютере. Для этого надо скопировать установщик на сменный носитель, например флэшку, и попытаться установить программу с нее. Если установка будет выполнена успешно, то работу по созданию установщика можно считать законченной.

ГЛАВА 11

Примеры программ

Экзаменатор

Тестирование широко применяется для оценки уровня знаний в учебных заведениях, при приеме на работу, для оценки квалификации персонала и т. п. Тест — это последовательность вопросов, на которые испытуемый должен ответить, выбрав правильный ответ из нескольких предложенных вариантов.

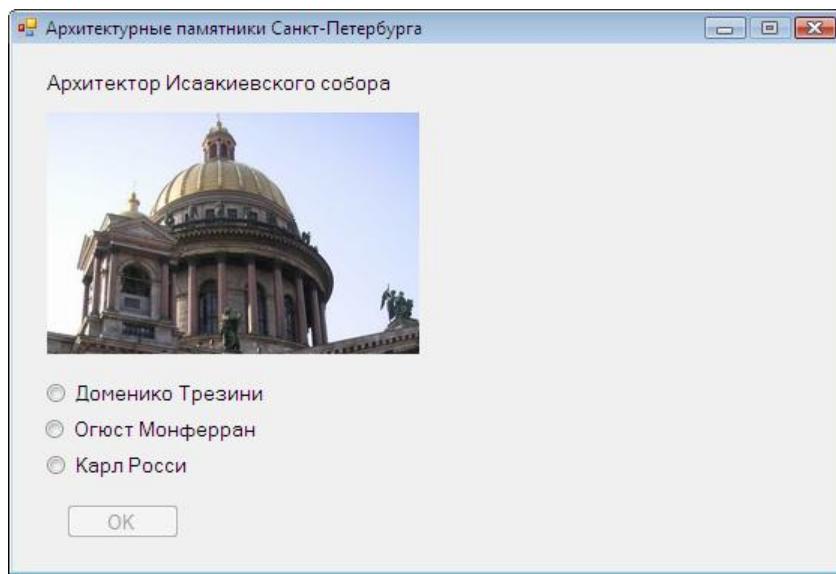


Рис. 11.1. Задача испытуемого — выбрать правильный ответ

Программа "Экзаменатор" позволяет автоматизировать процесс тестирования. В ее окне (рис. 11.1) отображается тест — последовательность вопросов, на которые испытуемый должен ответить путем выбора правильного ответа из нескольких предложенных вариантов.

Требования к программе

В результате анализа используемых на практике методик тестирования были сформулированы следующие требования к программе:

- ◆ программа должна быть универсальной, у пользователя должна быть возможность самостоятельной, без участия программиста, подготовки теста;
- ◆ программа должна работать с тестом произвольной длины, т. е. не должно быть ограничения на количество вопросов в teste;
- ◆ ответ на вопрос должен осуществляться путем выбора одного ответа из нескольких (не более четырех) вариантов;
- ◆ в программе должна быть заблокирована возможность возврата к предыдущему вопросу. Если вопрос предложен, то на него должен быть дан ответ;
- ◆ результат тестирования должен быть отнесен к одному из четырех уровней, например: "отлично", "хорошо", "удовлетворительно" или "плохо";
- ◆ вопрос может сопровождаться иллюстрацией.

Файл теста

Универсальность программы тестирования обеспечивается тем, что вопросы загружаются из XML-файла, имя которого указывается в команде запуска программы. Помимо вопросов (здесь и далее: вопрос — это собственно вопрос и несколько вариантов ответа) в файле теста находится информация, необходимая для выставления оценки.

Тест (узел верхнего уровня <test>) состоит из:

- ◆ заголовка;
- ◆ описания;
- ◆ раздела вопросов;
- ◆ раздела оценок.

Заголовок и описание представляют собой соответственно узлы <head> и <description>.

Вот пример заголовка и описания:

```
<head>
    Архитектурные памятники Санкт-Петербурга
</head>
<description>
    Сейчас Вам будут предложены вопросы по архитектуре Санкт-Петербурга.
    Задача — из предложенных вариантов ответа выбрать правильный.
</description>
```

За разделом описания следует раздел вопросов теста — узел <qw>, подузлы <q> которого представляют собой вопросы.

Узел <q> представляет собой вопрос, а его подузлы <a> — варианты ответа. Параметр text узла <q> задает текст вопроса, а параметр src — имя файла иллюстра-

ции. Узел `<a>` определяет текст варианта ответа, а параметр `right` указывает, является ли ответ правильным (значение "yes") или нет (значение "no").

Вот пример вопроса:

```
<q text="Архитектор Исаакиевского собора" src ="is.jpg" >
  <a right="no">Доменико Трезини</a>
  <a right="yes">Огюст Монферран</a>
  <a right="no">Карл Росси</a>
</q>
```

В приведенном примере вопрос "Архитектор Исаакиевского собора" сопровождается иллюстрацией, находящейся в файле `is.jpg`. К вопросу даны три варианта ответа, правильным является второй ответ (архитектор Исаакиевского собора — Огюст Монферран).

За разделом вопросов следует раздел оценок (узел `<levels>`), в котором указывается количество баллов (правильных ответов), необходимое для достижения уровня, и сообщение, информирующее испытуемого о достигнутом уровне (оценке). Каждому уровню соответствует узел `level`, параметр `score` которого задает количество баллов, необходимое для достижения уровня, параметр `text` — сообщение.

Вот пример раздела оценок:

```
<levels>
  <level score="2"
    text = "На все вопросы Вы ответили правильно. Оценка - ОТЛИЧНО."/>
  <level score="1"
    text = "На некоторые вопросы Вы ответили не правильно. Оценка - ХОРОШО."/>
  <level score="0"
    text = "Вы плохо подготовились к испытанию. Оценка - ПЛОХО!"/>
</levels>
```

В листинге 11.1 в качестве примера приведен файл теста "Архитектурные памятники Санкт-Петербурга".

Листинг 11.1. Файл теста (spb.xml)

```
<?xml version="1.0" encoding="Windows-1251"?>
<test>
  <head>
    Архитектурные памятники Санкт-Петербурга
  </head>
  <description>
    Сейчас Вам будут предложены вопросы по архитектуре Санкт-Петербурга.
    Задача — из предложенных вариантов ответа выбрать правильный.
  </description>
  <qw>
    <q text="Архитектор Исаакиевского собора" src ="is.jpg" >
      <a right="no">Доменико Трезини</a>
      <a right="yes">Огюст Монферран</a>
      <a right="no">Карл Росси</a>
    </q>
```

```
<q text="На фотографии" src ="marks.jpg">
  <a right="yes">Зимний дворец (Эрмитаж)</a>
  <a right="no">Марииинский дворец</a>
  <a right="no">Строгановский дворец</a>
</q>
</qw>
<levels>
  <level score="2" text = "На все вопросы Вы ответили правильно. Оценка –
отлично."/>
  <level score="1" text = "На некоторые вопросы Вы ответили не правильно.
Оценка – Хорошо."/>
  <level score="0" text = "Вы плохо подготовились к испытанию. Оценка –
плохо!"/>
</levels>
</test>
```

Файл теста можно подготовить в текстовом редакторе, который сохраняет "чистый" (без символов форматирования) текст, например в Блокноте. В момент записи текста на диск вместо стандартного расширения xml следует указать какое-либо другое, например ext (от англ. *examiner* — экзаменатор). Такое решение позволит настроить операционную систему так, что тест будет запускаться автоматически, в результате щелчка кнопкой мыши на имени файла теста.

Форма

Форма программы "Экзаменатор" приведена на рис. 11.2.

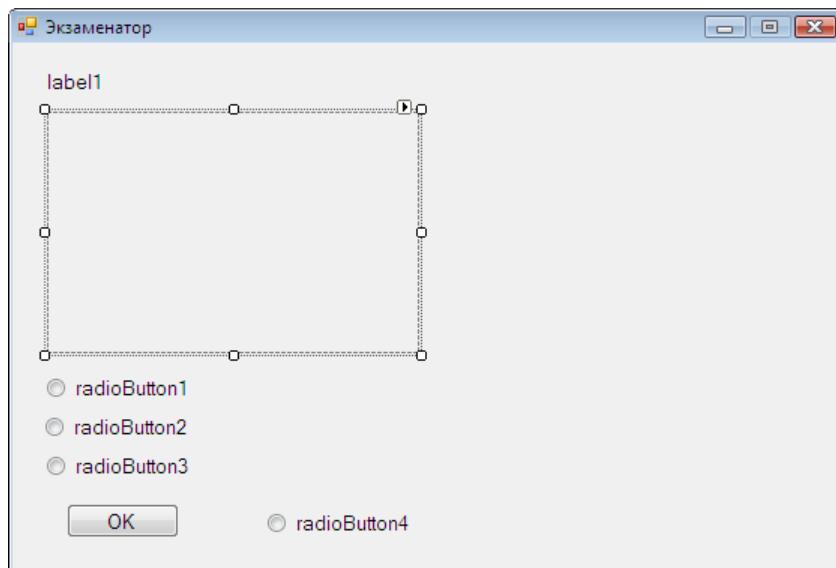


Рис. 11.2. Форма программы "Экзаменатор"

Вопрос отображается в поле `label1`, варианты ответа — в полях компонентов `radioButton1`—`radioButton3`. Эти же компоненты обеспечивают выбор ответа, а `radioButton4` (во время работы программы он не отображается) — сброс переключателей выбора ответа. Кнопка `button1` (она становится доступной только после выбора варианта ответа) обеспечивает переход к следующему вопросу. Обратите внимание, что текст на кнопке `button1` во время работы программы меняется. В начале и в конце работы программы на кнопке написано **OK**, а в процессе тестирования — **Дальше**. Если вопрос сопровождается иллюстрацией, то она отображается в поле компонента `pictureBox1`.

В табл. 11.1 приведены значения свойств формы, в табл. 11.2 — значения свойств компонентов.

Таблица 11.1. Значения свойств формы

Свойство	Значение
Text	Экзаменатор
FormBorderStyle	FixedSingle
MaximizeBox	False
StartPosition	CenterScreen

Таблица 11.2. Значения свойств компонентов

Компонент	Свойство	Значение
<code>label1</code>	AutoSize	True
	MaximumSize	500;0
<code>radioButton1</code>	Visible	False
	AutoSize	True
<code>radioButton2</code>	Visible	False
	AutoSize	True
<code>radioButton3</code>	Visible	False
	AutoSize	True
<code>radioButton4</code>	Visible	False
<code>pictureBox1</code>	Visible	False
	Size	248;160
	SizeMode	Zoom
<code>radioButton4</code>	Visible	False

Следует обратить внимание, что значения свойств, определяющих положение компонентов, предназначенных для отображения вопроса, альтернативных ответов и иллюстрации, вычисляются во время работы программы, после того как будет

прочитан очередной вопрос. Положение компонента `radioButton1` отсчитывается от нижней границы компонента `label1` или, если вопрос сопровождается иллюстрацией, от нижней границы компонента `pictureBox1`. Положение компонента `radioButton2` отсчитывается от нижней границы компонента `radioButton1`. Аналогичным образом вычисляется положение компонента `radioButton3`.

Доступ к файлу теста

Для того чтобы программа "Экзаменатор" была действительно универсальной, у пользователя должна быть возможность задать имя файла теста.

Обеспечить возможность настройки программы на работу с конкретным файлом можно несколькими способами. Например, имя файла теста можно указать в команде запуска программы.

Командная строка — это команда (строка), которую пользователь должен набрать в окне **Запуск программы**, для того чтобы запустить программу. В простейшем случае командная строка — это имя ехе-файла. В командной строке после имени ехе-файла можно указать дополнительную информацию, которую надо передать программе, например имя файла, с которым должна работать программа.

Параметры, указанные пользователем при запуске программы, передаются функции `main`. Для того чтобы функция `main` могла получить параметры командной строки, ее объявление должно выглядеть так:

```
static void Main(string[] args)
```

Как видно из объявления, параметр `args` — строковый массив. Элементы этого массива содержат параметры, указанные в команде запуска.

Функция `main` в свою очередь должна передать параметры командной строки конструктору формы. Как это делается, показано в листинге 11.2.

Конструктор формы программы "Экзаменатор" приведен в листинге 11.3. Обратите внимание, объявление конструктора соответствует инструкции вызова конструктора в файле `Program.cs`.

Листинг 11.2. "Экзаменатор" (Program.cs)

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

namespace WindowsFormsApplication1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
```

```
static void Main(string[] args)
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1(args)); // !!!
}
}
```

Листинг 11.3. "Экзаменатор" (Конструктор формы)

```
public Form1(string[] args)
{
    InitializeComponent();

    // имя файла теста должно быть указано
    // в качестве параметра команды запуска программы
    if (args.Length == 0)
    {
        label1.Text = "Файл теста необходимо указать " +
                      "в команде запуска программы.\n" +
                      "Например: 'exam economics.xml' " +
                      "или 'exam c:\\spb.xml'.";

        mode = 2;
        return;
    }

    // указано только имя файла?
    if (args[0].IndexOf(":") == -1)
    {
        fpath = Application.StartupPath + "\\";
        fname = args[0];
    }

    else
    {
        // указан путь к файлу теста
        fpath = args[0].Substring(0, args[0].LastIndexOf("\\") + 1);
        fname = args[0].Substring(args[0].LastIndexOf("\\") + 1);
    }

    try
    {
        // получаем доступ к xml-документу
        xmlReader = new System.Xml.XmlTextReader(fpath + fname);
        xmlReader.Read();
    }
```

```
mode = 0;
n    = 0;

// загрузить заголовок теста
this.showHead();

// загрузить описание теста
this.showDescription();
}

catch (Exception exc)
{
    label1.Text = "Ошибка доступа к файлу " + fpath + fname;
    mode = 2;
}
}
```

Как было сказано ранее, при запуске программы "Экзаменатор" из операционной системы имя файла теста надо указать в команде ее запуска. При запуске программы из Visual C# имя файла теста нужно ввести в поле **Command line arguments** вкладки **Debug** окна **Properties** (рис. 11.3), которое становится доступным в результате выбора в меню **Project** команды **Properties**.

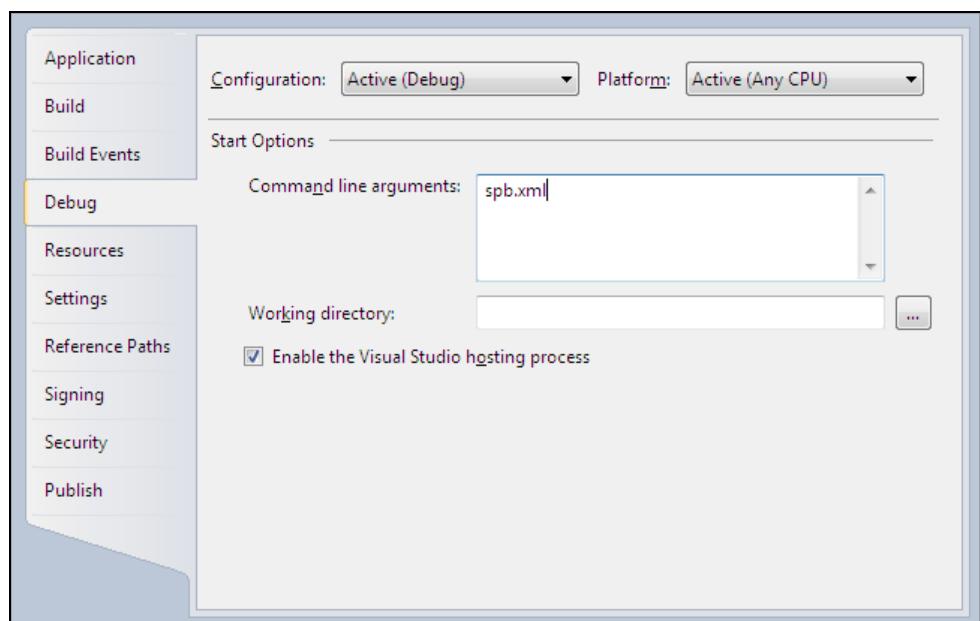


Рис. 11.3. Параметры командной строки
надо ввести в поле **Command line arguments**

Текст программы "Экзаменатор"

Модуль формы программы "Экзаменатор" приведен в листинге 11.4.

Листинг 11.4. "Экзаменатор" (Form1.cs)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        string fpath; // путь к файлу теста
        string fname; // файл теста

        // XmlReader обеспечивает чтение данных xml-файла
        System.Xml.XmlReader xmlReader;

        string qw; // вопрос

        // варианты ответа
        string[] answ = new string[3];

        string pic; // путь к файлу иллюстрации

        int right; // правильный ответ (номер)
        int otv; // выбранный ответ (номер)
        int n; // количество правильных ответов
        int nv; // общее количество вопросов
        int mode; // состояние программы:
                  // 0 - начало работы;
                  // 1 - тестирование;
                  // 2 - завершение работы

        // Конструктор формы.
        // args - параметры командной строки
        // (см. также Program.cs)
        public Form1(string[] args)
        {
            InitializeComponent();
        }
    }
}
```

```
// имя файла теста должно быть указано
// в качестве параметра команды запуска программы
if (args.Length == 0)
{
    label1.Text =
        "Файл теста необходимо указать " +
        "в команде запуска программы.\n" +
        "Например: 'exam economics.xml' " +
        "или 'exam c:\\spb.xml'.";
    mode = 2;
    return;
}

// указано только имя файла?
if (args[0].IndexOf(":") == -1)
{
    fpath = Application.StartupPath + "\\";
    fname = args[0];
}

else
{
    // указан путь к файлу теста
    fpath = args[0].Substring(0,args[0].LastIndexOf("\\")+1);
    fname = args[0].Substring(args[0].LastIndexOf("\\")+1);
}

try
{
    // получаем доступ к xml-документу
    xmlReader = new System.Xml.XmlTextReader(fpath + fname);
    xmlReader.Read();

    mode = 0;
    n = 0;

    // загрузить заголовок теста
    this.showHead();

    // загрузить описание теста
    this.showDescription();
}

catch(Exception exc)
{
    label1.Text = "Ошибка доступа к файлу " + fpath + fname;
    mode = 2;
}
```

```
// выводит название (заголовок) теста
private void showHead()
{
    // ищем узел <head>
    do xmlReader.Read();
    while(xmlReader.Name != "head");

    // считываем заголовок
    xmlReader.Read();

    // вывести название теста в заголовок окна
    this.Text = xmlReader.Value;

    // выходим из узла <head>
    xmlReader.Read();
}

// выводит описание теста
private void showDescription()
{
    // ищем узел <description>
    do
        xmlReader.Read();
    while(xmlReader.Name != "description");

    // считываем описание теста
    xmlReader.Read();

    // выводим описание теста
    label1.Text = xmlReader.Value;

    // выходим из узла <description>
    xmlReader.Read();

    // ищем узел вопросов <qw>
    do
        xmlReader.Read();
    while(xmlReader.Name != "qw");

    // входим внутрь узла
    xmlReader.Read();
}

// читает вопрос из файла теста
private Boolean getQw() {
    // считываем тег <q>
    xmlReader.Read();
```

```
if (xmlReader.Name == "q")
{
    // Здесь прочитан тег <q>, атрибут text которого содержит вопрос,
    // а атрибут src - имя файла иллюстрации.

    // Извлекаем значение атрибутов:
    qw = xmlReader.GetAttribute("text");
    pic = xmlReader.GetAttribute("src");
    if (!pic.Equals(string.Empty)) pic = fpath + pic;

    // Входим внутрь узла
    xmlReader.Read();
    int i = 0;

    // считываем данные узла вопроса <q>
    while (xmlReader.Name != "q")
    {
        xmlReader.Read();

        // варианты ответа
        if (xmlReader.Name == "a")
        {
            // запоминаем правильный ответ
            if (xmlReader.GetAttribute("right") == "yes")
                right = i;

            // считываем вариант ответа
            xmlReader.Read();
            if (i < 3) answ[i] = xmlReader.Value;

            // выходим из узла <a>
            xmlReader.Read();

            i++;
        }
    }

    // выходим из узла вопроса <q>
    xmlReader.Read();

    return true;
}
// если считанный тег не является
// тегом вопроса <q>
else
    return false;
}
```

```
// выводит вопрос и варианты ответа
private void showQw() {
    // выводим вопрос
    label1.Text = qw;

    // иллюстрация
    if (pic.Length != 0)
    {
        try
        {
            pictureBox1.Image = new Bitmap(pic);

            pictureBox1.Visible = true;

            radioButton1.Top = pictureBox1.Bottom + 16;
        }
        catch
        {
            if (pictureBox1.Visible)
                pictureBox1.Visible = false;

            label1.Text += "\n\n\nОшибка доступа к файлу " + pic + ".";
        }
    }
    else
    {
        if (pictureBox1.Visible)
            pictureBox1.Visible = false;

        radioButton1.Top = label1.Bottom;
    }

    // показать варианты ответа
    radioButton1.Text = answ[0];
    radioButton2.Top = radioButton1.Top + 24;;
    radioButton2.Text = answ[1];
    radioButton3.Top = radioButton2.Top + 24;;
    radioButton3.Text = answ[2];

    radioButton4.Checked = true;
    button1.Enabled = false;
}

// Щелчок на кнопке выбора ответа.
// Функция обрабатывает событие Click
// компонентов radioButton1 - radioButton3
```

```
private void radioButton1_Click(object sender, EventArgs e)
{
    if ((RadioButton)sender == radioButton1) otv = 0;
    if ((RadioButton)sender == radioButton2) otv = 1;
    if ((RadioButton)sender == radioButton3) otv = 2;

    button1.Enabled = true;
}

// щелчок на кнопке OK
private void button1_Click_1(object sender, EventArgs e)
{
    switch (mode)
    {
        case 0:           // начало работы программы
            radioButton1.Visible = true;
            radioButton2.Visible = true;
            radioButton3.Visible = true;

            this.getQw();
            this.showQw();

            mode = 1;

            button1.Enabled = false;
            radioButton4.Checked = true;
            break;

        case 1:
            nv++;

            // правильный ли ответ выбран
            if (otv == right) n++;

            if (this.getQw()) this.showQw();
            else {
                // больше вопросов нет
                radioButton1.Visible = false;
                radioButton2.Visible = false;
                radioButton3.Visible = false;

                pictureBox1.Visible = false;

                // обработка и вывод результата
                this.showLevel();

                // следующий щелчок на кнопке OK
                // закроет окно программы
            }
    }
}
```

```
        mode = 2;
    }
    break;
case 2: // завершение работы программы
    this.Close(); // закрыть окно
    break;
}

// выводит оценку
private void showLevel()
{
    // ищем узел <levels>
    do
        xmlReader.Read();
    while (xmlReader.Name != "levels");

    // входим внутрь узла
    xmlReader.Read();

    // читаем данные узла
    while (xmlReader.Name != "levels")
    {
        xmlReader.Read();

        if (xmlReader.Name == "level")
            // n - кол-во правильных ответов;
            // проверяем, попадаем ли в категорию
            if (n >= System.Convert.ToInt32(
                xmlReader.GetAttribute("score")))
                break;
    }

    label1.Text = "Тестирование завершено.\n" +
        "Всего вопросов: " + nv.ToString() + ". " +
        "Правильных ответов: " + n.ToString() + ".\n" +
        xmlReader.GetAttribute("text");
}
}
```

Работает программа "Экзаменатор" так. Конструктор формы проверяет, указан ли при запуске программы параметр — имя файла. Если параметр не указан (в этом случае размер массива `args` равен нулю), то в поле `label1` выводится сообщение о необходимости указать файл теста и переменной `mode` присваивается значение 2. Далее, т. к. значение `mode` равно 2, то в результате щелчка на кнопке **OK** работа программы заканчивается. Если параметр в команде запуска указан, то проверяет-

ся, указано ли полное имя файла (включая путь). Если указано полное имя файла, то в переменную `fpath` записывается путь к файлу теста. Делается это для того, чтобы в дальнейшем можно было получить доступ к файлам иллюстраций (по условию задачи, файлы иллюстраций должны находиться в том же каталоге, что и файл теста). Если имя файла краткое, то в переменную `fpath` записывается имя каталога, в котором находится exe-файл. Имя файла теста записывается в переменную `fname`. Далее выполняется проверка, существует ли указанный файл теста. Если файла теста нет, то выводится сообщение об ошибке и программа завершает работу. Конструктор формы вызывает функции `showHead` и `showDescription`, которые соответственно читают из файла заголовок и описание теста (узлы `<head>` и `<description>`) и отображают их: заголовок — в заголовке формы, описание — в поле `label1`. В результате в окне программы отображаются описание теста и кнопка **OK**. После того как испытуемый прочтет информацию о teste и сделает щелчок на кнопке **OK**, начинается процесс тестирования.

Основную работу выполняет функция обработки события на кнопке `button1`. Следует обратить внимание, что эта кнопка используется для активизации процесса тестирования, перехода к следующему вопросу (после выбора варианта ответа) и завершения работы программы. Какое из перечисленных действий будет выполнено, определяет значение переменной `mode`. В начале работы программы значение переменной равно нулю. Поэтому в результате первого щелчка на кнопке выполняется та часть программы, которая обеспечивает отображение первого вопроса и записывает в переменную `mode` единицу. В процессе тестирования значение в переменной `mode` равно единице. Поэтому функция обработки события `click` увеличивает на единицу счетчик правильных ответов (если выбран правильный ответ) и вызывает функцию `getQw`, которая считывает из файла очередной вопрос и, затем — функцию `showQw`, которая выводит его на экран. Если текущий вопрос последний (в этом случае функция `getQw` возвращает `False`), то вызывается функция `showLevel`, которая выводит на экран результат тестирования и в переменную `mode` записывает двойку. В результате, после следующего щелчка на кнопке `button1`, программа завершает работу.

Читает очередной вопрос из файла функция `getQw`. Сначала она читает узел `qw` и в переменную `qw` записывает текст вопроса — значение атрибута `text`. В переменную `pic` записывается значение атрибута `src` — имя файла иллюстрации. Далее из подузлов `q` читаются вопросы. В процессе чтения контролируется значение атрибута `right`. Если значение равно `yes`, то в переменной `right` фиксируется номер правильного ответа (порядковый номер подузла).

Функция `showQw` выводит вопрос на экран. Необходимо обратить внимание на то, что положение области отображения первого ответа (компонент `radioButton1`) отсчитывается от нижней границы области отображения иллюстрации (компонента `pictureBox1`) или от нижней границы области отображения вопроса (компонента `label1`). Если к вопросу есть иллюстрация, то она отображается, и свойство `top` компонента `radioButton1` устанавливается так, чтобы он располагался немного ниже нижней границы иллюстрации. Если иллюстрации нет, то положение компонента `radioButton1` отсчитывается от нижней границы компонента `label1`.

Обработку события Click на переключателях radioButton1—radioButton3 выполняет одна, общая для всех компонентов, функция radioButton1_Click. Она фиксирует в переменной `otv` номер выбранного ответа (номер переключателя) и делает доступной кнопку перехода к следующему вопросу.

Функция `showLevel` (она запускается в результате щелчка на кнопке `button1`, если значение переменной `mode` равно 2) читает информацию об уровнях оценки (подузлы `<level>` узла `<levels>`) и выводит на экран результат тестирования.

Запуск программы

Программа "Экзаменатор" получает имя файла теста из командной строки. Поэтому чтобы "запустить тест", пользователь должен в окне **Выполнить** набрать имя программы тестирования и указать файл теста (рис. 11.4), что не совсем удобно.

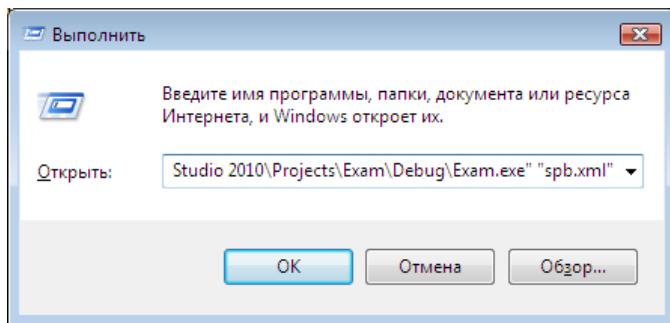


Рис. 11.4. Запуск программы "Экзаменатор" из Windows

Чтобы облегчить жизнь пользователю, можно настроить операционную систему так, чтобы программа тестирования запускалась в результате щелчка на значке файла теста. Для этого сначала надо изменить расширение имени файла теста, например на `exm` (от англ. *examiner* — экзаменатор). Затем нужно раскрыть папку, в которой находится какой-либо файл теста (`exm`-файл), сделать щелчок правой кнопкой на значке файла и в появившемся контекстном меню выбрать команду **Открыть с помощью**. В появившемся окне **Выбор программы** нужно сделать щелчок на кнопке **Обзор**, раскрыть папку, в которой находится программа "Экзаменатор", и указать `exe`-файл. После этого надо установить флажок **Использовать выбранную программу для всех файлов такого типа**. Вид окна **Выбор программы** после выполнения всех перечисленных действий приведен на рис. 11.5.

В результате описанных действий в реестр операционной системы будет внесена информация о том, что файлы с расширением `exm` надо открывать с помощью программы "Экзаменатор" (имя файла, на котором сделан щелчок, передается программе как параметр).

Если для развертывания приложения предполагается использовать технологию ClickOnce (см. главу 10), то задачу привязки файла теста к программе "Экзаменатор" можно возложить на установщик. Для этого в окне **Project Properties** надо

раскрыть вкладку **Publish**, сделать щелчок на кнопке **Options** и в разделе **File Associations** задать расширение, описание и значок файла теста (рис. 11.6).

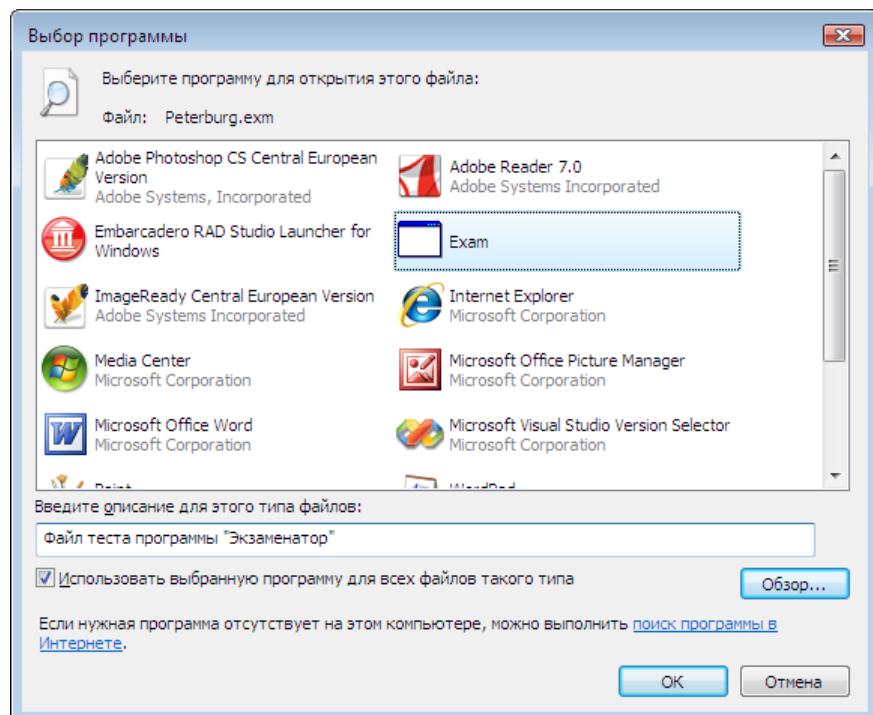


Рис. 11.5. Теперь файлы с расширением exm будут открывать "Экзаменатор"

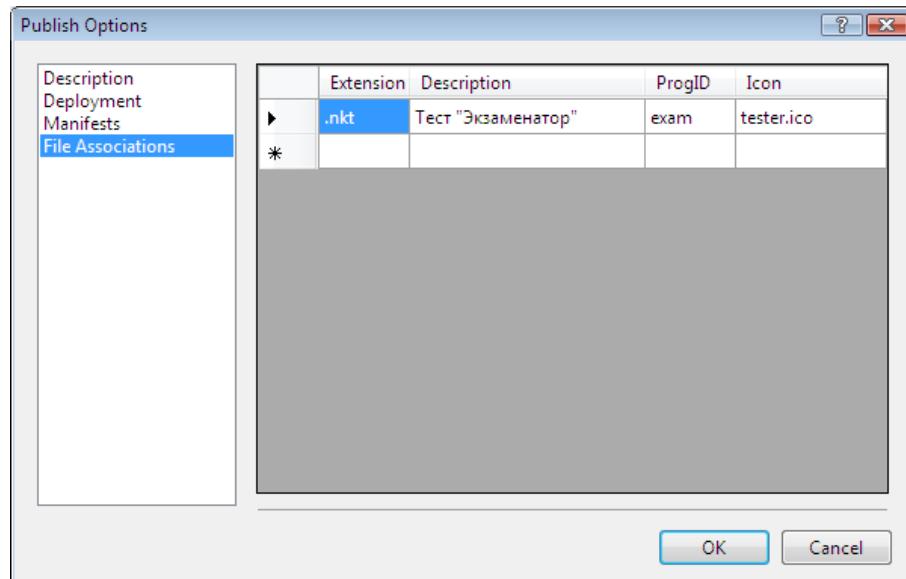


Рис. 11.6. Связывание типа файла с приложением

Команда запуска приложения, установленного с использованием технологии ClickOnce deployed, представляет собой URL-ссылку. Щелчок на значке приложения, находящемся, например, на рабочем столе, "открывает" ссылку, в результате чего приложение запускается. Поэтому конструктор формы следует изменить — заменить инструкции доступа к параметрам командной строки на инструкции доступа к Web-параметрам (листинг 11.5).

Листинг 11.5. Конструктор ClickOnce deployed-приложения

```
public Form1(string[] args)
{
    InitializeComponent();

    // Имя файла теста должно быть указано
    // в качестве параметра в команды запуска программы.
    // Доступ к параметрам ClickOnce deployed-приложения

    string cmdLine;
    try
    {
        cmdLine = AppDomain.CurrentDomain.SetupInformation.
            ActivationArguments.ActivationData[0];
    }
    catch
    {
        // программа запущена щелчком на ее значке
        label1.Text = "Для запуска теста, сделайте щелчок на nkt-файле";
        mode = 2;
        return;
    }

    // указано только имя файла?
    if (cmdLine.IndexOf(":") == -1)
    {
        fpath = Application.StartupPath + "\\";
        fname = cmdLine;
    }

    else
    {
        // указан путь к файлу теста
        fpath = cmdLine.Substring(0, cmdLine.LastIndexOf("\\") + 1);
        fname = cmdLine.Substring(cmdLine.LastIndexOf("\\") + 1);
    }

    try
    {
        // получаем доступ к xml-документу
```

```

xmlReader = new System.Xml.XmlTextReader(fpath + fname);
xmlReader.Read();

mode = 0;
n = 0;

// загрузить заголовок теста
this.showHead();

// загрузить описание теста
this.showDescription();
}

catch (Exception exc)
{
    label1.Text = "Ошибка доступа к файлу " + fpath + fname;
    mode = 2;
}
}
}

```

Сапер

Всем, кто работает с операционной системой Windows, хорошо знакома игра "Сапер". В этом разделе рассматривается аналогичная программа.

Пример окна программы в конце игры (после того как игрок открыл клетку, в которой находится мина) приведен на рис. 11.7.

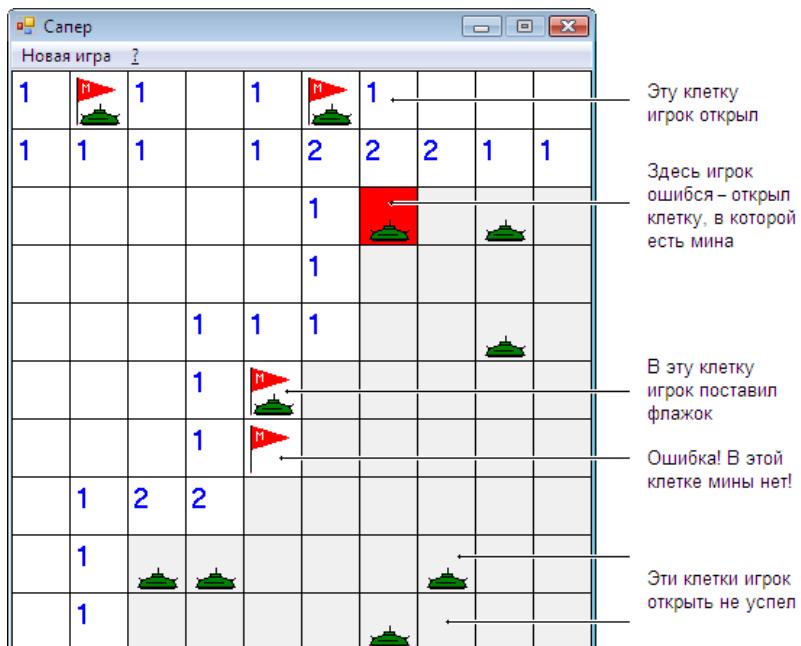


Рис. 11.7. Окно программы "Сапер"

Правила и представление данных

Игровое поле состоит из клеток, в каждой из которых может быть мина. Задача игрока — найти все мины и пометить их флагами.

Используя кнопки мыши, игрок может открыть клетку или поставить в нее флагок, указав тем самым, что в клетке находится мина. Клетка открывается щелчком левой кнопки мыши, флагок ставится щелчком правой. Если в клетке, которую открыл игрок, есть мина, то происходит взрыв (сапер ошибся, а он, как известно, ошибается только один раз), и игра заканчивается. Если в клетке мины нет, то в этой клетке появляется число, соответствующее количеству мин, находящихся в соседних клетках. Анализируя информацию о количестве мин в клетках, соседних с уже открытыми, игрок может обнаружить и пометить флагками все мины. Ограничений на количество клеток, помеченных флагками, нет. Однако для завершения игры (выигрыша) флагки должны быть установлены только в тех клетках, в которых есть мины. Ошибочно установленный флагок можно убрать, щелкнув правой кнопкой мыши в клетке, в которой он находится.

В программе игровое поле представлено массивом $N + 2$ на $M + 2$, где $N \times M$ — размер игрового поля. Элементы массива с номерами строк от 1 до N и номерами столбцов от 1 до M соответствуют клеткам игрового поля (рис. 11.8), первые и последние столбцы и строки соответствуют границе игрового поля.

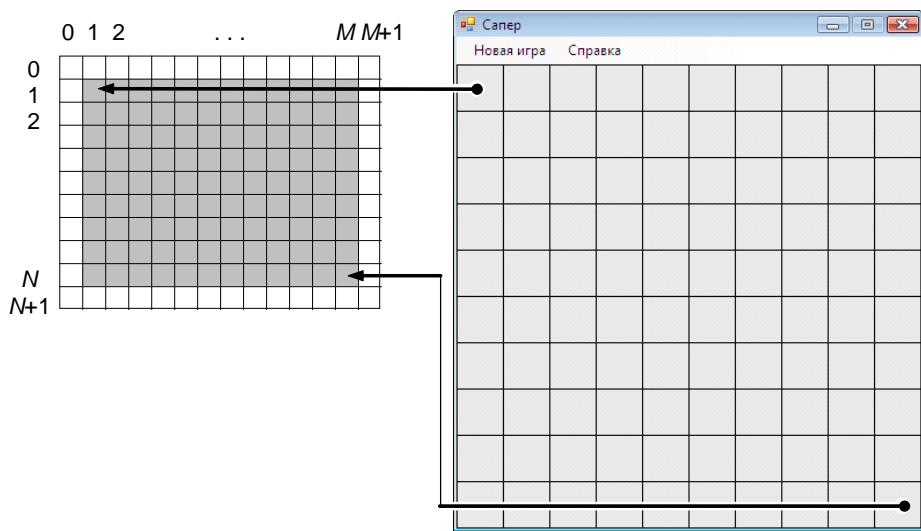


Рис. 11.8. Клетке игрового поля соответствует элемент массива

В начале игры каждый элемент массива, соответствующий клеткам игрового поля, может содержать число от 0 до 9. Ноль соответствует пустой клетке, рядом с которой нет мин. Клеткам, в которых нет мин, но рядом с которыми мины есть, соответствуют числа от 1 до 8. Элементы массива, соответствующие клеткам, в которых находятся мины, имеют значение 9.

Элементы массива, соответствующие границе поля, содержат -3.

В качестве примера на рис. 11.9 изображен массив, соответствующий состоянию поля в начале игры.

-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3
-3	9	1	0	0	0	0	0	0	0	0	0	0	-3
-3	1	1	0	0	0	0	0	0	0	0	0	0	-3
-3	1	2	2	1	0	0	0	0	1	1	1	1	-3
-3	1	9	9	1	0	0	0	0	2	9	2	-3	
-3	1	2	2	1	0	0	0	0	2	9	3	-3	
-3	0	0	0	0	0	0	0	0	2	3	9	-3	
-3	0	1	2	2	1	0	0	0	1	9	2	-3	
-3	0	2	9	9	1	0	0	0	1	1	1	1	-3
-3	0	2	9	3	1	0	0	0	0	0	0	0	-3
-3	0	1	1	1	0	0	0	0	0	0	0	0	-3
-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3	-3

Рис. 11.9. Массив в начале игры

В процессе игры состояние игрового поля меняется (игрок открывает клетки и ставит флаги), и соответственно меняются значения элементов массива. Если игрок поставил в клетку флагок, то значение соответствующего элемента массива увеличивается на 100. Например, если флагок поставлен правильно — в клетку, в которой есть мина, то значение соответствующего элемента массива станет 109. Если флагок поставлен ошибочно, например в пустую клетку, элемент массива будет содержать число 100. Если игрок открыл клетку, то значение элемента массива увеличивается на 200. Такой способ кодирования позволяет сохранить информацию об исходном состоянии клетки.

Форма

Главная форма игры "Сапер" приведена на рис. 11.10, значения ее свойств — в табл. 11.3. Компонент `panel1` используется в качестве поверхности, на которой формируется графика. Чтобы компонент `panel1` занимал всю рабочую область формы, свойству `Dock` следует присвоить значение `Fill`.

Таблица 11.3. Значения свойств стартовой формы

Свойство	Значение
Text	Сапер
FormBorderStyle	FixedSingle
MaximizeBox	False
StartPosition	CenterScreen

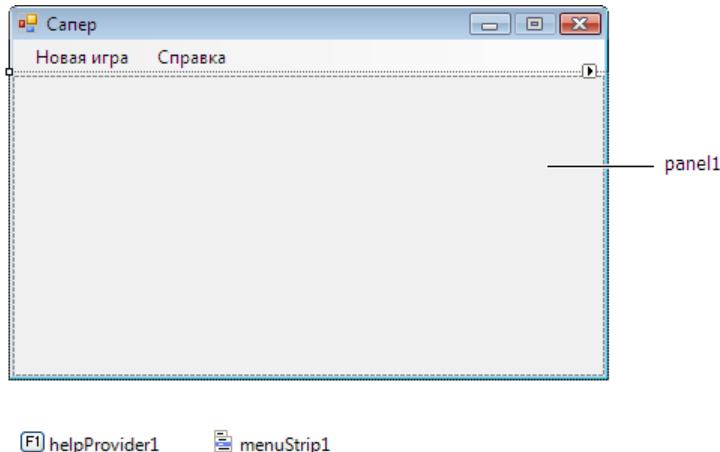


Рис. 11.10. Главная форма программы "Сапер"

Следует обратить внимание на то, что размер формы не соответствует размеру игрового поля. Нужный размер формы будет установлен во время работы программы. Делает это конструктор формы, который на основе информации о размере клеток и их количестве по вертикали и горизонтали устанавливает значение свойства `ClientSize`, определяющего размер клиентской (рабочей) области окна программы.

Структура меню программы приведена на рис. 11.11.

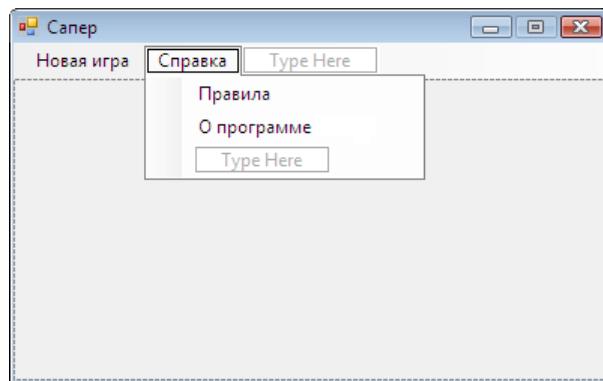


Рис. 11.11. Структура меню программы "Сапер"

После того как будет сформирована структура меню, надо создать процедуры обработки события `Click` для команд **Новая игра**, **Правила** и **О программе**.

Игровое поле

На разных этапах игры игровое поле выглядит по-разному. Сначала поле просто разделено на клетки. Во время игры, в результате щелчка правой кнопкой мыши, в клетке появляется флагок. Щелчок левой кнопкой открывает клетку — в ней по-

является цифра (количество мин в соседних клетках) или мина (игра на этом заканчивается).

Начало игры

В начале игры программа должна расставить мины и для каждой клетки поля подсчитать, сколько мин находится в соседних клетках. Функция newGame (ее текст приведен в листинге 11.6), решает эту задачу.

Листинг 11.6. Функция newGame

```
// новая игра
private void newGame() {
    int row, col;      // индексы клетки
    int n = 0;         // количество поставленных мин
    int k;             // количество мин в соседних клетках

    // очистить поле
    for(row = 1; row <= MR; row++)
        for(col = 1; col <= MC; col++)
            Pole[row,col] = 0;

    // инициализация генератора случайных чисел
    Random rnd = new Random();

    // расставим мины
    do
    {
        row = rnd.Next(MR) + 1;
        col = rnd.Next(MC) + 1;

        if (Pole[row,col] != 9)
        {
            Pole[row,col] = 9;
            n++;
        }
    }
    while (n != NM);

    // для каждой клетки вычислим количество мин в соседних клетках
    for(row = 1; row <= MR; row++)
        for(col = 1; col <= MC; col++)
            if (Pole[row,col] != 9)
            {
                k = 0;

                if (Pole[row-1,col-1] == 9) k++;
                if (Pole[row-1,col] == 9) k++;
                if (Pole[row+1,col-1] == 9) k++;
                if (Pole[row+1,col] == 9) k++;
                if (Pole[row-1,col+1] == 9) k++;
                if (Pole[row+1,col+1] == 9) k++;
            }
    }
}
```

```

    if (Pole[row-1,col+1] == 9) k++;
    if (Pole[row,col-1] == 9) k++;
    if (Pole[row,col+1] == 9) k++;
    if (Pole[row+1,col-1] == 9) k++;
    if (Pole[row+1,col] == 9) k++;
    if (Pole[row+1,col+1] == 9) k++;

    Pole[row,col] = k;
}

status = 0;          // начало игры
nMin   = 0;          // нет обнаруженных мин
nFlag  = 0;          // нет поставленных флагов
}

```

Функция `showPole` выводит изображение поля: последовательно, одну за другой рисует клетки. Вывод изображения отдельной клетки выполняет функция `kletka`. Функции `showPole` и `kletka` приведены в листинге 11.7. Функция `kletka` используется для вывода изображения поля в начале игры, во время игры и в ее конце. В начале игры (значение параметра `status` равно в этом случае нулю) функция выводит только контур клетки, во время игры — количество мин, находящихся в соседних клетках, или флагок (если клетка была открыта щелчком правой кнопкой мыши), а в конце игры отображает исходное состояние клетки (если клетка не была открыта) или результат действия игрока. Информацию о фазе игры функция `kletka` получает через параметр `status`.

Листинг 11.7. Функции `showPole` и `kletka`

```

// рисует поле
private void showPole(Graphics g, int status) {
    for(int row = 1; row <= MR; row++)
        for(int col = 1; col <= MC; col++)
            this.kletka(g, row, col, status);
}

// рисует клетку
private void kletka(Graphics g, int row, int col, int status) {
    int x, y; // координаты левого верхнего угла клетки

    x = (col - 1) * W + 1;
    y = (row-1)* H + 1;

    // не открытые клетки - серые
    if (Pole[row,col] < 100)
        g.FillRectangle(SystemBrushes.ControlLight, x-1, y-1, W, H);

```

```
// открытые или помеченные клетки
if (Pole[row,col] >= 100) {

    // открываем клетку, открытые - белые
    if (Pole[row,col] != 109)
        g.FillRectangle(Brushes.White, x-1, y-1, W, H);
    else
        // на этой мине подорвались!
        g.FillRectangle(Brushes.Red, x-1, y-1, W, H);

    // если в соседних клетках есть мины,
    // указываем их количество
    if ((Pole[row,col] >= 101) && (Pole[row,col] <= 108))
        g.DrawString((Pole[row,col]-100).ToString(),
            new Font("Tahoma", 10, System.Drawing.FontStyle.Regular),
            Brushes.Blue, x+3, y+2);
}

// в клетке поставлен флаг
if (Pole[row,col] >= 200)
    this.flag(g, x, y);

// рисуем границу клетки
g.DrawRectangle(Pens.Black, x-1, y-1, W, H);

// если игра завершена (status = 2), показываем мины
if ((status == 2) && ((Pole[row,col] % 10) == 9))
    this.mina(g, x, y);
}
```

Игра

Во время игры программа воспринимает нажатия кнопок мыши и в соответствии с правилами игры открывает клетки или ставит в клетки флаги.

Основную работу выполняет функция обработки события `MouseClick` (ее текст приведен в листинге 11.8). Функция через параметр `e` получает координаты точки окна (панели), в которой игрок щелкнул кнопкой мыши, а также информацию о том, какая кнопка была нажата. Сначала процедура преобразует координаты точки, в которой игрок нажал кнопку мыши, в координаты клетки игрового поля. Затем делает необходимые изменения в массиве `Pole` и, если нажата правая кнопка, вызывает функцию `flag`, которая рисует в клетке флагок. Если нажата левая кнопка в клетке, в которой нет мины, то эта клетка открывается — на экране появляется ее содержимое. Если левая кнопка нажата в клетке, в которой есть мина, то вызывается процедура `showPole`, которая показывает все мины, в том числе и те, которые игрок не успел найти.

Листинг 11.8. Обработка события MouseClick на поверхности игрового поля

```
// щелчок кнопкой в клетке игрового поля
private void panel1_MouseClick(object sender, MouseEventArgs e)
{
    // игра завершена
    if (status == 2) return;

    // первый щелчок
    if (status == 0) status = 1;

    // преобразуем координаты мыши в индексы клетки поля,
    // в которой был сделан щелчок;
    // (e.X, e.Y) - координаты точки формы,
    // в которой была нажата кнопка мыши;
    int row = (int) (e.Y/H) + 1,
        col = (int) (e.X/W) + 1;

    // координаты области вывода
    int x = (col-1)* W + 1,
        y = (row-1)* H + 1;

    // щелчок левой кнопки мыши
    if (e.Button == MouseButtons.Left)
    {
        // открыта клетка, в которой есть мина
        if (Pole[row,col] == 9)
        {
            Pole[row,col] += 100;

            // игра закончена
            status = 2;

            // перерисовать форму
            this.panel1.Invalidate();
        }
        else
            if (Pole[row,col] < 9)
                this.open(row,col);
    }

    // щелчок правой кнопки мыши
    if (e.Button == MouseButtons.Right) {

        // в клетке не было флага, ставим его
        if (Pole[row,col] <= 9) {
            nFlag += 1;
```

```

if (Pole[row,col] == 9)
    nMin += 1;

Pole[row,col] += 200;

if ((nMin == NM) && (nFlag == NM)) {
    // игра закончена
    status = 2;

    // перерисовываем все игровое поле
    this.Invalidate();
}

else
    // перерисовываем только клетку
    this.kletka(g, row, col, status);
}

else
    // в клетке был поставлен флаг,
    // повторный щелчок правой кнопки мыши
    // убирает его и закрывает клетку
    if (Pole[row,col] >= 200)
    {
        nFlag -= 1;
        Pole[row,col] -= 200;

        // перерисовываем клетку
        this.kletka(g, row, col, status);
    }
}

```

Функция `flag` (листинг 11.9) рисует флагок. Флажок (рис. 11.12) состоит из четырех примитивов: линии (древко), замкнутого контура (флаг) и ломаной линии (буква "M"). Процедура рисует флагок, используя метод базовой точки, т. е. координаты всех точек, определяющих положение элементов рисунка, отсчитываются от базовой точки.

Функция `mina` (листинг 11.10) рисует мину (рис. 11.13), которая состоит из восьми примитивов: два прямоугольника и сектор образуют корпус мины, остальные элементы рисунка — линии ("усы" и полоски на корпусе).

Обеим функциям в качестве параметров передаются координаты базовой точки, от которой надо рисовать.

Листинг 11.9. Функция flag

```
// рисует флаг  
private void flag(Graphics q, int x, int y)
```

```

{
    Point[] p = new Point[3];
    Point[] m = new Point[5];

    // флагок
    p[0].X = x+4;    p[0].Y = y+4;
    p[1].X = x+30;   p[1].Y = y+12;
    p[2].X = x+4;    p[2].Y = y+20;
    g.FillPolygon(Brushes.Red, p);

    // деревко
    g.DrawLine(Pens.Black,
        x+4, y+4, x+4, y+35);

    // буква "M" на флагке
    m[0].X = x+8;    m[0].Y = y+14;
    m[1].X = x+8;    m[1].Y = y+8;
    m[2].X = x+10;   m[2].Y = y+10;
    m[3].X = x+12;   m[3].Y = y+8;
    m[4].X = x+12;   m[4].Y = y+14;
    g.DrawLines(Pens.White, m);
}

```

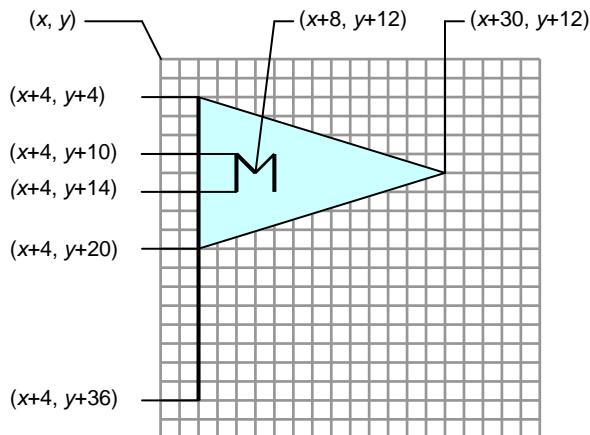


Рис. 11.12. Флажок

Листинг 11.10. Функция mina

```

// рисует мину
private void mina(Graphics g, int x, int y)
{
    // корпус
    g.FillRectangle(Brushes.Green, x+16, y+26, 8, 4);

```

```

g.FillRectangle(Brushes.Green, x+8, y+30, 24, 4);
g.DrawPie(Pens.Black, x+6, y+28, 28, 16, 0, -180);
g.FillPie(Brushes.Green, x+6, y+28, 28, 16, 0, -180);

// полоса на корпусе
g.DrawLine(Pens.Black, x+12, y+32, x+28, y+32);

// вертикальный "ус"
g.DrawLine(Pens.Black, x+20, y+22, x+20, y+26);

// боковые "усы"
g.DrawLine(Pens.Black, x+8, y+30, x+6, y+28);
g.DrawLine(Pens.Black, x+32, y+30, x+34, y+28);
}

```

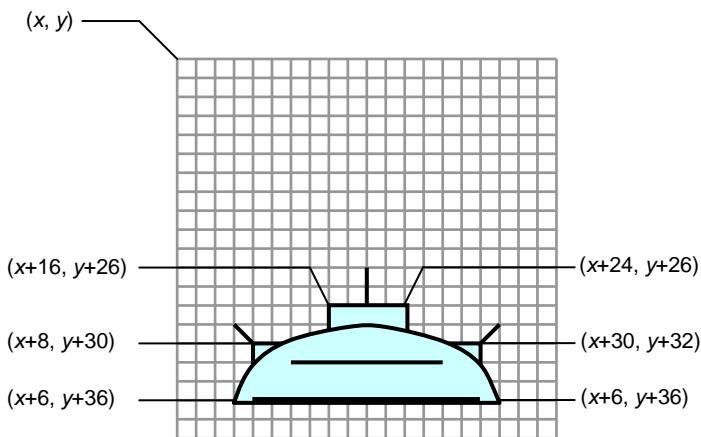


Рис. 11.13. Мина

Справочная информация

В результате выбора в меню **Справка** команды **Правила** появляется окно справочной информации (рис. 11.14).

Активизирует процесс отображения справочной информации функция обработки события `Click` (листинг 11.11) на элементе меню **Правила**.

Листинг 11.11. Отображение справочной информации

```

// выбор в меню Справка команды Правила игры
private void правилаToolStripMenuItem_Click(object sender, EventArgs e)
{
    Help.ShowHelp(this, helpProvider1.HelpNamespace, "saper_02.htm");
}

```

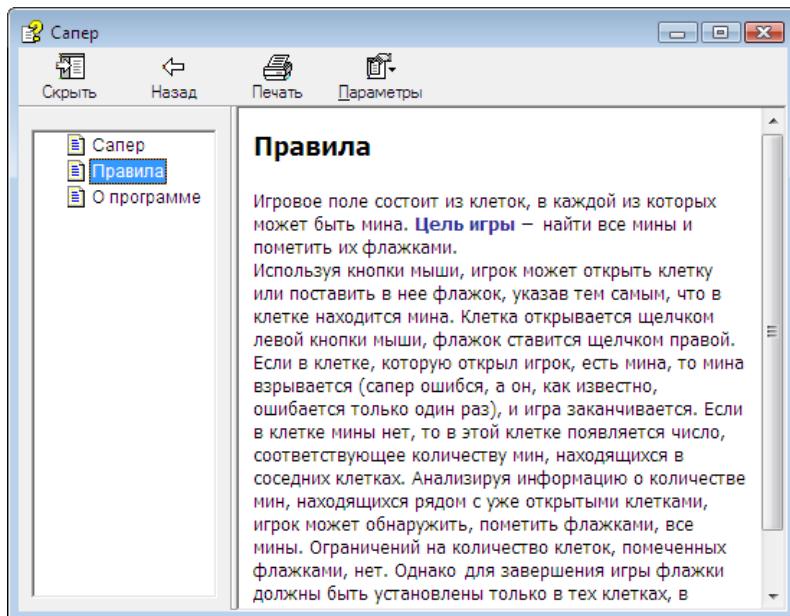


Рис. 11.14. Окно справочной информации программы "Сапер"

Информация о программе

При выборе в меню **Справка** команды **О программе** на экране появляется одноименное диалоговое окно (рис. 11.15).

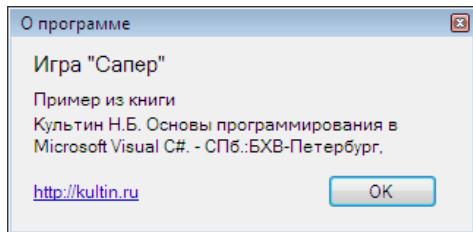


Рис. 11.15. Окно О программе

Чтобы программа во время своей работы могла вывести на экран окно, отличное от главного (стартового), в проект нужно добавить форму: в меню **Project** выбрать команду **Add New Item**, в появившемся окне выбрать **Windows Form** и в поле **Name** ввести имя формы. В результате выполнения этих действий в проект будет добавлена новая форма.

Форма **О программе** (**Form2**) приведена на рис. 11.16, значения свойств формы и компонентов — в табл. 11.4 и 11.5.

Вывод окна **О программе** выполняет функция обработки события **Click**, которое происходит в результате выбора в меню **Справка** команды **О программе** (листинг 11.12).

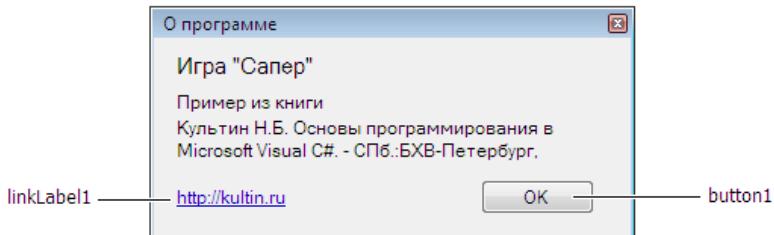


Рис. 11.16. Форма О программе

Таблица 11.4. Значения свойств формы О программе

Свойство	Значение
Text	О программе
FormBorderStyle	FixedToolWindow
ShowTaskbar	False
StartPosition	CenterParent

Таблица 11.5. Значения свойств компонентов формы О программе

Компонент	Свойство	Значение
linkLabel1	Text	http://kultin.ru
button1	DialogResult	OK

Непосредственно отображение окна **О программе** выполняет метод `ShowDialog`, который выводит окно, как *модальный* диалог. Модальный диалог перехватывает все события, адресованные другим окнам приложения, в том числе и главному. Таким образом, пока модальный диалог находится на экране, продолжить работу с приложением, которое вывело модальный диалог, нельзя.

Листинг 11.12. Отображение окна О программе

```
// выбор в меню Справка команды О программе
private void oПрограммеToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 about = new Form2();
    about.ShowDialog();
}
```

В окне **О программе** есть Web-ссылка. Предполагается, что в результате щелчка на ссылке в окне браузера будет открыта указанная страница. Запуск браузера обеспечивает метод (функция) `start` (листинг 11.13). Эта функция достаточно универсальна. Она обеспечивает выполнение операций с файлами и протоколами, тип

которых известен операционной системе. В данном случае необходимо открыть Web-страницу, поэтому в качестве параметра функции передается адрес страницы (URL-ссылка). Следует обратить внимание, что функция `Start` не запускает конкретную программу, а информирует операционную систему о необходимости открыть указанный файл. Поэтому в результате щелчка на ссылке будет запущен браузер, установленный на компьютере пользователя.

Листинг 11.13. Щелчок на Web-ссылке

```
// щелчок на Web-ссылке
private void linkLabel1_LinkClicked(object sender,
                                    LinkLabelLinkClickedEventArgs e)
{
    string webRef = linkLabel1.Text;
    System.Diagnostics.Process.Start(webRef);
}
```

Окно **О программе** закрывается в результате щелчка на кнопке **OK**. Здесь необходимо обратить внимание, в программе нет функции обработки события `Click` на кнопке **OK**. Однако в результате щелчка на кнопке **OK** окно программы закрывается. Это происходит потому, что свойству `DialogResult` кнопки `button1` присвоено значение `OK` (по умолчанию значение этого свойства равно `None`).

Текст программы

Полный текст программы "Сапер" приведен в листингах 11.14 (модуль главной формы) и 11.15 (модуль формы **О программе**).

Листинг 11.14. Модуль главной формы

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        private const int
            MR = 10, // кол-во клеток по вертикали
            MC = 10, // кол-во клеток по горизонтали
```

```
NM = 10, // кол-во мин
W = 40, // ширина клетки
H = 40; // высота клетки

// игровое (минное) поле
private int[,] Pole = new int[MR + 2, MC + 2];
// значение элемента массива:
// 0..8 - количество мин в соседних клетках
// 9 - в клетке мина
// 100..109 - клетка открыта
// 200..209 - в клетку поставлен флаг

private int nMin; // кол-во найденных мин
private int nFlag; // кол-во поставленных флагов

// статус игры
private int status;
// 0 - начало игры,
// 1 - игра,
// 2 - результат

// графическая поверхность формы
private System.Drawing.Graphics g;

public Form1()
{
    InitializeComponent();

    // В неотображаемые элементы массива, соответствующие
    // клеткам границы игрового поля, запишем число -3.
    // Это значение используется процедурой open()
    // для завершения рекурсивного процесса открытия
    // соседних пустых клеток
    for(int row = 0; row <= MR+1; row++)
    {
        Pole[row, 0] = -3;
        Pole[row, MC+1] = -3;
    }

    for(int col = 0; col <= MC+1; col++)
    {
        Pole[0,col] = -3;
        Pole[MR+1,col] = -3;
    }

    // устанавливаем размер формы в соответствии
    // с размером игрового поля
    this.ClientSize =
        new Size(W*MC + 1, H*MR + menuStrip1.Height + 1);
}
```

```
newGame() ; // новая игра

// графическая поверхность
g = panel1.CreateGraphics();
}

// новая игра
private void newGame() {
    int row, col;      // индексы клетки
    int n = 0;         // количество поставленных мин
    int k;             // количество мин в соседних клетках

    // очистить поле
    for(row = 1; row <= MR; row++)
        for(col = 1; col <= MC; col++)
            Pole[row,col] = 0;

    // инициализация генератора случайных чисел
    Random rnd = new Random();

    // расставим мины
    do
    {
        row = rnd.Next(MR) + 1;
        col = rnd.Next(MC) + 1;

        if (Pole[row,col] != 9)
        {
            Pole[row,col] = 9;
            n++;
        }
    }
    while (n != NM);

    // для каждой клетки вычислим кол-во мин в соседних клетках
    for(row = 1; row <= MR; row++)
        for(col = 1; col <= MC; col++)
            if (Pole[row,col] != 9)
            {
                k = 0;

                if (Pole[row-1,col-1] == 9) k++;
                if (Pole[row-1,col] == 9) k++;
                if (Pole[row-1,col+1] == 9) k++;
                if (Pole[row,col-1] == 9) k++;
                if (Pole[row,col+1] == 9) k++;
                if (Pole[row+1,col-1] == 9) k++;
            }
    }
}
```

```

if (Pole[row+1,col] == 9) k++;
if (Pole[row+1,col+1] == 9) k++;

Pole[row,col] = k;
}

status = 0;           // начало игры
nMin   = 0;           // нет обнаруженных мин
nFlag  = 0;           // нет поставленных флагов
}

// рисует поле
private void showPole(Graphics g, int status) {
    for(int row = 1; row <= MR; row++)
        for(int col = 1; col <= MC; col++)
            this.kletka(g, row, col, status);
}

// рисует клетку
private void kletka(Graphics g, int row, int col, int status) {
    int x, y; // координаты левого верхнего угла клетки

    x = (col - 1) * W + 1;
    y = (row-1)* H + 1;

    // не открытые клетки - серые
    if (Pole[row,col] < 100)
        g.FillRectangle(SystemBrushes.ControlLight, x-1, y-1, W, H);

    // открытые или помеченные клетки
    if (Pole[row,col] >= 100) {

        // открываем клетку, открытые - белые
        if (Pole[row,col] != 109)
            g.FillRectangle(Brushes.White, x-1, y-1, W, H);
        else
            // на этой мине подорвались!
            g.FillRectangle(Brushes.Red, x-1, y-1, W, H);

        // если в соседних клетках есть мины,
        // указываем их количество
        if ((Pole[row,col] >= 101) && (Pole[row,col] <= 108))
            g.DrawString((Pole[row,col]-100).ToString(),
                new Font("Tahoma", 10, System.Drawing.FontStyle.Regular),
                Brushes.Blue, x+3, y+2);
    }
}

```

```
// в клетке поставлен флаг
if (Pole[row,col] >= 200)
    this.flag(g, x, y);

// рисуем границу клетки
g.DrawRectangle(Pens.Black, x-1, y-1, W, H);

// если игра завершена (status = 2), показываем мины
if ((status == 2) && ((Pole[row,col] % 10) == 9))
    this.mina(g, x, y);
}

// открывает текущую и все соседние с ней клетки, в которых нет мин
private void open(int row, int col)
{
    // координаты области вывода
    int x = (col-1)* W + 1,
        y = (row-1)* H + 1;

    if (Pole[row,col] == 0)
    {
        Pole[row,col] = 100;

        // отобразить содержимое клетки
        this.kletka(g, row, col, status);

        // открыть примыкающие клетки слева, справа, сверху, снизу
        this.open(row, col-1);
        this.open(row-1, col);
        this.open(row, col+1);
        this.open(row+1, col);

        // примыкающие диагонально
        this.open(row-1,col-1);
        this.open(row-1,col+1);
        this.open(row+1,col-1);
        this.open(row+1,col+1);
    }
    else
        if ((Pole[row,col] < 100) && (Pole[row,col] != -3))
    {
        Pole[row,col] += 100;

        // отобразить содержимое клетки
        this.kletka(g, row, col, status);
    }
}
```

```
// рисует мину
private void mina(Graphics g, int x, int y)
{
    // корпус
    g.FillRectangle(Brushes.Green, x+16, y+26, 8, 4);
    g.FillRectangle(Brushes.Green, x+8, y+30, 24, 4);
    g.DrawPie(Pens.Black, x+6, y+28, 28, 16, 0, -180);
    g.FillPie(Brushes.Green, x+6, y+28, 28, 16, 0, -180);

    // полоса на корпусе
    g.DrawLine(Pens.Black, x+12, y+32, x+28, y+32);

    // вертикальный "ус"
    g.DrawLine(Pens.Black, x+20, y+22, x+20, y+26);

    // боковые "усы"
    g.DrawLine(Pens.Black, x+8, y+30, x+6, y+28);
    g.DrawLine(Pens.Black, x+32, y+30, x+34, y+28);
}

// рисует флаг
private void flag(Graphics g, int x, int y)
{
    Point[] p = new Point[3];
    Point[] m = new Point[5];

    // флагок
    p[0].X = x+4;    p[0].Y = y+4;
    p[1].X = x+30;   p[1].Y = y+12;
    p[2].X = x+4;    p[2].Y = y+20;
    g.FillPolygon(Brushes.Red, p);

    // древко
    g.DrawLine(Pens.Black, x+4, y+4, x+4, y+35);

    // буква "M" на флагке
    m[0].X = x+8;    m[0].Y = y+14;
    m[1].X = x+8;    m[1].Y = y+8;
    m[2].X = x+10;   m[2].Y = y+10;
    m[3].X = x+12;   m[3].Y = y+8;
    m[4].X = x+12;   m[4].Y = y+14;
    g.DrawLines(Pens.White, m);
}

// щелчок кнопкой в клетке игрового поля
private void panel1_MouseClick(object sender, MouseEventArgs e)
```

```
{  
    // игра завершена  
    if (status == 2) return;  
  
    // первый щелчок  
    if (status == 0) status = 1;  
  
    // преобразуем координаты мыши в индексы  
    // клетки поля, в которой был сделан щелчок;  
    // (e.X, e.Y) - координаты точки формы,  
    // в которой была нажата кнопка мыши;  
    int row = (int)(e.Y/H) + 1,  
        col = (int)(e.X/W) + 1;  
  
    // координаты области вывода  
    int x = (col-1)* W + 1,  
        y = (row-1)* H + 1;  
  
    // щелчок левой кнопки мыши  
    if (e.Button == MouseButtons.Left)  
    {  
        // открыта клетка, в которой есть мина  
        if (Pole[row,col] == 9)  
        {  
            Pole[row,col] += 100;  
  
            // игра закончена  
            status = 2;  
  
            // перерисовать форму  
            this.panel1.Invalidate();  
        }  
        else  
        if (Pole[row,col] < 9)  
            this.open(row,col);  
    }  
  
    // щелчок правой кнопки мыши  
    if (e.Button == MouseButtons.Right) {  
  
        // в клетке не было флага, ставим его  
        if (Pole[row,col] <= 9) {  
            nFlag += 1;  
  
            if (Pole[row,col] == 9)  
                nMin += 1;  
        }  
    }  
}
```

```
Pole[row,col] += 200;

if ((nMin == NM) && (nFlag == NM)) {
    // игра закончена
    status = 2;

    // перерисовываем все игровое поле
    this.Invalidate();
}
else
    // перерисовываем только клетку
    this.kletka(g, row, col, status);
}

else
    // в клетке был поставлен флаг,
    // повторный щелчок правой кнопки мыши
    // убирает его и закрывает клетку
    if (Pole[row,col] >= 200)
    {
        nFlag -= 1;
        Pole[row,col] -= 200;

        // перерисовываем клетку
        this.kletka(g, row, col, status);
    }
}

// команда Новая игра
private void новаяИграToolStripMenuItem_Click(object sender, EventArgs e)
{
    newGame();
    showPole(g, status);
}

// обработка события Paint панели
private void panel1_Paint(object sender, PaintEventArgs e)
{
    showPole(g, status);
}

// выбор в меню Справка команды О программе
private void оПрограммеToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 about = new Form2();
    about.ShowDialog();
}
```

```
// выбор в меню Справка команды Правила игры
private void правилаToolStripMenuItem_Click(object sender, EventArgs e)
{
    Help.ShowHelp(this, helpProvider1.HelpNamespace, "saper_02.htm");
}

}
```

Листинг 11.15. Модуль формы О программе

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication2
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        // щелчок на Web-ссылке
        private void linkLabel1_LinkClicked(object sender,
                                             LinkLabelLinkClickedEventArgs e)
        {
            string webRef = linkLabel1.Text;
            System.Diagnostics.Process.Start(webRef);
        }
    }
}
```




ГЛАВА 12

Краткий справочник

Форма

Свойства формы (объекта WinForm) приведены в табл. 12.1.

Таблица 12.1. Свойства формы

Свойство	Описание
Name	Имя формы
Text	Текст в заголовке
Size	Размер формы. Уточняющее свойство Width определяет ширину, свойство Height — высоту
StartPosition	Положение формы в момент первого появления на экране. Форма может находиться в центре экрана (CenterScreen), в центре родительского окна (CenterParent). Если значение свойства равно Manual, то положение формы определяется значением свойства Location
Location	Положение формы на экране. Расстояние от верхней границы формы до верхней границы экрана задает уточняющее свойство Y, расстояние от левой границы формы до левой границы экрана — уточняющее свойство X
FormBorderStyle	Тип формы (границы). Форма может представлять собой обычное окно (Sizable), окно фиксированного размера (FixedSingle, Fixed3D), диалог (FixedDialog) или окно без кнопок Свернуть и Развернуть (SizeableToolWindow, FixedToolWindow). Если свойству присвоить значение None, у окна не будет заголовка и границы
ControlBox	Управляет отображением системного меню и кнопок управления окном. Если значение свойства равно False, то в заголовке окна кнопка системного меню, а также кнопки Свернуть , Развернуть , Закрыть не отображаются
MaximizeBox	Кнопка Развернуть . Если значение свойства равно False, то находящаяся в заголовке окна кнопка Развернуть недоступна
MinimizeBox	Кнопка Свернуть . Если значение свойства равно False, то находящаяся в заголовке окна кнопка Свернуть недоступна

Таблица 12.1 (окончание)

Свойство	Описание
Icon	Значок в заголовке окна
Font	Шрифт, используемый по умолчанию компонентами, находящимися на поверхности формы. Изменение значения свойства приводит к автоматическому изменению значения свойства Font всех компонентов формы (при условии, что значение свойства компонента не было задано явно)
ForeColor	Цвет, наследуемый компонентами формы и используемый ими для отображения текста. Изменение значения свойства приводит к автоматическому изменению соответствующего свойства всех компонентов формы (при условии, что значение свойства Font компонента не было задано явно)
BackColor	Цвет фона. Можно задать явно (выбрать на вкладке Custom или Web) или указать элемент цветовой схемы (выбрать на вкладке System)
Opacity	Степень прозрачности формы. Форма может быть непрозрачной (100%) или прозрачной. Если значение свойства меньше 100%, то сквозь форму видна поверхность, на которой она отображается

Компоненты

В этом разделе кратко описаны основные (базовые) компоненты. Подробное описание этих и других компонентов можно найти в справочной системе.

Button

Компонент Button представляет собой командную кнопку. Свойства компонента приведены в табл. 12.2.

Таблица 12.2. Свойства компонента *Button*

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
Text	Текст на кнопке
TextAlign	Положение текста на кнопке. Текст может располагаться в центре кнопки (<i>MiddleCenter</i>), быть прижат к левой (<i>MiddleLeft</i>) или правой (<i>MiddleRight</i>) границе. Можно задать и другие способы размещения надписи (<i>TopLeft</i> , <i>TopCenter</i> , <i>TopRight</i> , <i>BottomLeft</i> , <i>BottomCenter</i> , <i>BottomRight</i>)
FlatStyle	Стиль. Кнопка может быть стандартной (<i>Standard</i>), плоской (<i>Flat</i>) или "всплывающей" (<i>PopUp</i>)
Location	Положение кнопки на поверхности формы. Уточняющее свойство <i>X</i> определяет расстояние от левой границы кнопки до левой границы формы, уточняющее свойство <i>Y</i> — от верхней границы кнопки до верхней границы клиентской области формы (нижней границы заголовка)

Таблица 12.2 (окончание)

Свойство	Описание
Size	Размер кнопки
Font	Шрифт, используемый для отображения текста на кнопке
ForeColor	Цвет текста, отображаемого на кнопке
Enabled	Признак доступности кнопки. Кнопка доступна, если значение свойства равно <code>True</code> , и недоступна, если значение свойства равно <code>False</code> (в этом случае нажать кнопку нельзя, событие <code>Click</code> в результате щелчка на ней не возникает)
Visible	Позволяет скрыть кнопку (<code>False</code>) или сделать ее видимой (<code>True</code>)
Cursor	Вид указателя мыши при позиционировании указателя на кнопке
Image	Картинка на поверхности кнопки. Рекомендуется использовать gif-файл, в котором определен прозрачный цвет
ImageAlign	Положение картинки на кнопке. Картина может располагаться в центре (<code>MiddleCenter</code>), быть прижата к левой (<code>MiddleLeft</code>) или правой (<code>MiddleRight</code>) границе. Можно задать и другие способы размещения картинки на кнопке (<code>TopLeft</code> , <code>TopCenter</code> , <code>TopRight</code> , <code>BottomLeft</code> , <code>BottomCenter</code> , <code>BottomRight</code>)
ImageList	Набор изображений, из которых может быть выбрано то, которое будет отображаться на поверхности кнопки. Представляет собой объект типа <code>ImageList</code> . Чтобы задать значение свойства, в форму приложения нужно добавить компонент <code>ImageList</code>
ImageIndex	Номер (индекс) изображения из набора <code>ImageList</code> , которое отображается на кнопке
ToolTip	Подсказка, появляющаяся рядом с указателем мыши при его позиционировании на кнопке. Чтобы свойство стало доступно, в форму приложения нужно добавить компонент <code>ToolTip</code>

ComboBox

Компонент `ComboBox` представляет собой комбинацию поля редактирования и списка, что дает возможность ввести данные путем набора на клавиатуре или выбором из списка. Свойства компонента приведены в табл. 12.3.

Таблица 12.3. Свойства компонента `ComboBox`

Свойство	Описание
<code>DropDownStyle</code>	Вид компонента: <code>DropDown</code> — поле ввода и раскрывающийся список; <code>Simple</code> — поле ввода со списком; <code>DropDownList</code> — раскрывающийся список
<code>Text</code>	Текст, находящийся в поле редактирования (для компонентов типа <code>DropDown</code> и <code>Simple</code>)
<code>Items</code>	Элементы списка — коллекция строк

Таблица 12.3 (окончание)

Свойство	Описание
Items->Count	Количество элементов списка
SelectedIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно -1
Sorted	Признак автоматической сортировки (True) списка после добавления очередного элемента
MaxDropDownItems	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше, чем MaxDropDownItems, то появляется вертикальная полоса прокрутки
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа DropDownList) или с учетом (для компонента типа Simple) размера области списка или области ввода
DropDownWidth	Ширина области списка
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

ContextMenuStrip

Компонент ContextMenuStrip представляет собой контекстное меню — список команд, который отображается в результате щелчка правой кнопкой мыши на форме или в поле компонента. Команды меню определяют значение свойства Items, представляющего собой коллекцию объектов MenuItem. Свойства объекта MenuItem приведены в табл. 12.4. Чтобы задать контекстное меню компонента или формы, нужно указать имя контекстного меню (компоненты ContextMenuStrip) в качестве значения свойства ContextMenuStrip.

Таблица 12.4. Свойства объекта MenuItem

Свойство	Описание
Text	Команда
Enabled	Признак доступности команды. Если значение свойства равно False, то команда недоступна (название команды отображается серым цветом)
Image	Картинка, которая отображается в строке команды
Checked	Признак того, что элемент меню выбран. Если значение свойства равно True, то элемент считается выбранным и помечается галочкой или (если значение свойства RadioCheck равно True) точкой. Свойство Checked обычно используется для тех элементов меню, которые предназначены для отображения параметров
RadioCheck	Признак того, что для индикации состояния свойства Checked используется точка (True), а не галочка (False)

CheckBox

Компонент CheckBox представляет собой флажок, который может находиться в одном из двух состояний: выбранном или невыбранном (часто вместо "выбранный" говорят "установленный", а вместо "невыбранный" — "сброшенный"). Свойства компонента CheckBox приведены в табл. 12.5.

Таблица 12.5. Свойства компонента CheckBox

Свойство	Описание
Text	Текст, отображаемый справа от флашка
Checked	Состояние флашка. Если флашок выбран (в поле компонента отображается галочка), то значение свойства равно <code>True</code> . Если флашок сброшен (галочка не отображается), то значение свойства равно <code>False</code>
TextAlign	Положение текста в поле отображения текста. Текст может располагаться в центре поля (<code>MiddleCenter</code>), быть прижат к левой (<code>MiddleLeft</code>) или правой (<code>MiddleRight</code>) границе. Можно задать и другие способы размещения текста надписи (<code>TopLeft</code> , <code>TopCenter</code> , <code>TopRight</code> , <code>BottomLeft</code> , <code>BottomCenter</code> , <code>BottomRight</code>)
CheckAlign	Положение флашка в поле компонента. Флашок может быть прижат к левой верхней границе (<code>TopLeft</code>), прижат к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (<code>MiddleLeft</code>). Есть и другие варианты размещения флашка в поле компонента
Enabled	Управляет доступностью компонента. Позволяет сделать флашок недоступным (<code>False</code>)
Visible	Управляет видимостью компонента. Позволяет скрыть, сделать невидимым (<code>False</code>) флашок
AutoCheck	Определяет, должно ли автоматически изменяться состояние флашка в результате щелчка на его изображении. По умолчанию значение равно <code>True</code>
FlatStyle	Стиль (вид) флашка. Флашок может быть обычным (<code>Standard</code>), плоским (<code>Flat</code>) или "всплывающим" (<code>PopUp</code>). Стиль определяет поведение флашка при позиционировании указателя мыши на его изображении
Appearance	Определяет вид флашка. Флашок может выглядеть обычным образом (<code>Normal</code>) или как кнопка (<code>Button</code>)
Image	Картина, которая отображается в поле компонента
ImageAlign	Положение картинки в поле компонента. Картина может располагаться в центре (<code>MiddleCenter</code>), быть прижата к левой (<code>MiddleLeft</code>) или правой (<code>MiddleRight</code>) границе. Можно задать и другие способы размещения картинки в поле компонента (<code>TopLeft</code> , <code>TopCenter</code> , <code>TopRight</code> , <code>BottomLeft</code> , <code>BottomCenter</code> , <code>BottomRight</code>)
ImageList	Набор картинок, используемых для обозначения различных состояний флашка. Представляет собой объект типа <code>ImageList</code> . Чтобы задать значение свойства, в форму приложения следует добавить компонент <code>ImageList</code>
ImageIndex	Номер (индекс) картинки из набора <code>ImageList</code> , которая отображается в поле компонента

CheckedListBox

Компонент `CheckedListBox` представляет собой список, перед каждым элементом которого находится флашок `CheckBox`. Свойства компонента `CheckedListBox` приведены в табл. 12.6.

Таблица 12.6. Свойства компонента `CheckedListBox`

Свойство	Описание
<code>Items</code>	Элементы списка — коллекция строк
<code>Items.Count</code>	Количество элементов списка
<code>Sorted</code>	Признак необходимости автоматической сортировки (<code>True</code>) списка после добавления очередного элемента
<code>CheckOnClick</code>	Способ пометки элемента списка. Если значение свойства равно <code>False</code> , то первый щелчок выделяет элемент списка (строку), а второй устанавливает в выбранное состояние флашок. Если значение свойства равно <code>True</code> , то щелчок на элементе списка выделяет элемент и устанавливает во включенное состояние флашок
<code>CheckedItems</code>	Элементы, выбранные в списке
<code>CheckedItems->Count</code>	Количество выбранных элементов
<code>CheckedIndices</code>	Коллекция, элементы которой содержат номера выбранных элементов списка
<code>MultiColumn</code>	Признак необходимости отображать список в несколько колонок. Число отображаемых колонок зависит от количества элементов и размера области отображения списка
<code>Location</code>	Положение компонента на поверхности формы
<code>Size</code>	Размер компонента без (для компонентов типа <code>DropDown</code> и <code>DropDownList</code>) или с учетом (для компонента типа <code>Simple</code>) размера области списка или области ввода
<code>Font</code>	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

GroupBox

Компонент `GroupBox` представляет собой контейнер для других компонентов. Обычно он используется для объединения в группы компонентов `RadioButton` по функциональному признаку. Свойства компонента `GroupBox` приведены в табл. 12.7.

Таблица 12.7. Свойства компонента `GroupBox`

Свойство	Описание
<code>Text</code>	Заголовок — текст, поясняющий назначение компонентов, которые находятся в поле компонента <code>GroupBox</code>

Таблица 12.7 (окончание)

Свойство	Описание
Enabled	Позволяет управлять доступом к компонентам, находящимся в поле (на поверхности) компонента <code>GroupBox</code> . Если значение свойства равно <code>False</code> , то все находящиеся в поле <code>GroupBox</code> компоненты недоступны
Visible	Позволяет скрыть (сделать невидимым) компонент <code>GroupBox</code> и все компоненты, которые находятся на его поверхности

ImageList

Компонент `ImageList` представляет собой коллекцию изображений и может использоваться другими компонентами (например, `Button` или `ToolBar`) как источник иллюстраций. Компонент является невизуальным, т. е. он не отображается в окне программы во время ее работы. Во время создания формы компонент отображается в нижней части окна редактора формы. Свойства компонента `ImageList` приведены в табл. 12.8.

Таблица 12.8. Свойства компонента `ImageList`

Свойство	Описание
<code>Images</code>	Коллекция изображений (объектов <code>Bitmap</code>)
<code>ImageSize</code>	Размер изображений коллекции. Уточняющее свойство <code>Width</code> определяет ширину изображений, <code>Height</code> — высоту
<code>TransparentColor</code>	Прозрачный цвет. Участки изображения, окрашенные этим цветом, не отображаются
<code>ColorDepth</code>	Глубина цвета — число байтов, используемых для кодирования цвета точки (пикселя)

Label

Компонент `Label` предназначен для отображения текстовой информации. Задать текст, отображаемый в поле компонента, можно как во время разработки формы, так и во время работы программы, присвоив нужное значение свойству `Text`. Свойства компонента приведены в табл. 12.9.

Таблица 12.9. Свойства компонента `Label`

Свойство	Описание
<code>Name</code>	Имя компонента. Используется в программе для доступа к свойствам компонента
<code>Text</code>	Отображаемый текст
<code>Location</code>	Положение компонента на поверхности формы

Таблица 12.9 (окончание)

Свойство	Описание
AutoSize	Признак автоматического изменения размера компонента. Если значение свойства равно <code>True</code> , то при изменении значения свойства <code>Text</code> (или <code>Font</code>) автоматически изменяется размер компонента
Size	Размер компонента (области отображения текста). Определяет (если значение свойства <code>AutoSize</code> равно <code>False</code>) размер компонента (области отображения текста)
MaximumSize	Если значение свойства <code>AutoSize</code> равно <code>True</code> , то задает максимально допустимый (максимально возможный) размер компонента (области отображения текста). Свойство <code>MaximumSize.Width</code> задает максимально возможную ширину области, свойство <code>MaximumSize.Height</code> — высоту
Font	Шрифт, используемый для отображения текста
ForeColor	Цвет текста, отображаемого в поле компонента
BackColor	Цвет закраски области вывода текста
TextAlign	Способ выравнивания (расположения) текста в поле компонента. Всего существует девять способов расположения текста. На практике наиболее часто используют выравнивание по левой верхней границе (<code>TopLeft</code>), посередине (<code>TopCenter</code>) и по центру (<code>MiddleCenter</code>)
BorderStyle	Вид рамки (границы) компонента. По умолчанию граница вокруг поля <code>Label</code> отсутствует (значение свойства равно <code>None</code>). Граница компонента может быть обычной (<code>Fixed3D</code>) или тонкой (<code>FixedSingle</code>)

ListBox

Компонент `ListBox` представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 12.10.

Таблица 12.10. Свойства компонента `ListBox`

Свойство	Описание
Items	Элементы списка — коллекция строк
Items.Count	Количество элементов списка
SelectedIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно <code>-1</code>
Sorted	Признак необходимости автоматической сортировки (<code>True</code>) списка после добавления очередного элемента
SelectionMode	Определяет режим выбора элементов списка: <code>One</code> — только один элемент; <code>MultiSimple</code> — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка; <code>MultiExtended</code> — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка при нажатой клавише <code><Ctrl></code> , или выделить диапазон, щелкнув при нажатой клавише <code><Shift></code> на первом и последнем элементах диапазона

Таблица 12.10 (окончание)

Свойство	Описание
MultiColumn	Признак необходимости отображать список в несколько колонок. Число отображаемых колонок зависит от количества элементов и размера области отображения списка
Location	Положение компонента на поверхности формы
Size	Размер компонента без учета (для компонентов типа <code>DropDown</code> и <code>DropDownList</code>) или с учетом (для компонента типа <code>Simple</code>) размера области списка или области ввода
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

MenuStrip

Компонент `MenuStrip` представляет собой главное меню программы. И сами меню, и команды, образующие меню, — это объекты `ToolStripMenuItem`. Свойства объекта `ToolStripMenuItem` приведены в табл. 12.11.

Таблица 12.11. Свойства объекта `ToolStripMenuItem`

Свойство	Описание
Text	Название меню (команды)
Enabled	Признак доступности меню (команды). Если значение свойства равно <code>False</code> , то меню (или команда) недоступно
Image	Картинка, отображаемая рядом с названием меню или команды. В качестве картинки следует использовать png-иллюстрацию с прозрачным фоном
Checked	Признак того, что элемент меню выбран. Если значение свойства равно <code>True</code> , то элемент помечается галочкой (если для него не задана картинка). Свойство <code>Checked</code> обычно используется для тех элементов меню, которые применяются для отображения параметров
Shortcut	Свойство определяет функциональную клавишу (или комбинацию клавиш), нажатие которой активизирует выполнение команды
ShowShortcut	Если значение свойства равно <code>True</code> и задано значение свойства <code>Shortcut</code> , то после названия команды отображается название функциональной клавиши, нажатие которой активизирует команду

NotifyIcon

Компонент `NotifyIcon` представляет собой значок, который отображается в системной области панели задач. Обычно он используется для изображения программ, работающих в фоновом режиме. При позиционировании указателя мыши на значке, как правило, появляется подсказка, а в результате щелчка правой кнопки — контекстное меню, команды которого позволяют управлять работой программы. Свойства компонента приведены в табл. 12.12.

Таблица 12.12. Свойства компонента *NotifyIcon*

Свойство	Описание
Icon	Значок, который отображается на панели задач
Text	Подсказка (обычно название программы), которая отображается рядом с указателем мыши при позиционировании указателя на находящемся на панели задач значке
ContextMenuStrip	Ссылка на компонент ContextMenuStrip, обеспечивающий отображение контекстного меню
Visible	Свойство позволяет скрыть (<i>False</i>) значок (убрать с панели задач) или сделать его видимым

NumericUpDown

Компонент *NumericUpDown* предназначен для ввода числовых данных. Данные в поле редактирования можно ввести путем набора на клавиатуре или при помощи кнопок **Увеличить** и **Уменьшить**, которые находятся справа от поля редактирования. Свойства компонента *NumericUpDown* приведены в табл. 12.13.

Таблица 12.13. Свойства компонента *NumericUpDown*

Свойство	Описание
Value	Значение, соответствующее содержимому поля редактирования
Maximum	Максимально возможное значение, которое можно ввести в поле компонента
Minimum	Минимально возможное значение, которое можно ввести в поле компонента
Increment	Величина, на которую увеличивается или уменьшается значение свойства Value при каждом щелчке мышью на кнопках Увеличить или Уменьшить
TextAlign	Расположение текста в поле редактирования (<i>Left</i> — прижат к левому краю; <i>Center</i> — в центре; <i>Right</i> — прижат к правому краю)

OpenFileDialog

Компонент *OpenFileDialog* представляет собой стандартное диалоговое окно **Открыть**. Свойства компонента приведены в табл. 12.14.

Таблица 12.14. Свойства компонента *OpenFileDialog*

Свойство	Описание
Title	Текст в заголовке окна. Если значение свойства не задано, то в заголовке отображается текст Открыть

Таблица 12.14 (окончание)

Свойство	Описание
Filter	Свойство задает описание и фильтр (маску) имени файла. В списке отображаются только те файлы, имена которых соответствуют указанной маске. Описание отображается в поле Тип файла . Например, значение Текст*.txt указывает, что в списке файлов нужно отобразить только файлы с расширением txt
FilterIndex	Если фильтр состоит из нескольких элементов (например, Текст*.txt Все файлы *.*), то значение свойства задает фильтр, который используется в момент появления диалога на экране
FileName	Имя файла, введенное пользователем или выбранное в списке файлов
InitialDirectory	Каталог, содержимое которого отображается при появлении диалога на экране
RestoreDirectory	Признак необходимости отображать содержимое каталога, указанного в свойстве InitialDirectory , при каждом появлении окна. Если значение свойства равно False , то при следующем появлении окна на экране отображается содержимое каталога, выбранного пользователем в предыдущий раз

Panel

Компонент **Panel** представляет собой контейнер для других компонентов и позволяет легко управлять компонентами, которые находятся на панели. Например, для того чтобы сделать недоступными компоненты, находящиеся на панели, достаточно присвоить значение **False** свойству **Enabled** панели. Свойства компонента **Panel** приведены в табл. 12.15.

Таблица 12.15. Свойства компонента *Panel*

Свойство	Описание
Dock	Определяет границу формы, к которой привязана (прикреплена) панель. Панель может быть прикреплена к левой (Left), правой (Right), верхней (Top) или нижней (Bottom) границе формы
BorderStyle	Вид границы панели: FixedSingle — рамка; Fixed3D — объемная граница; None — граница не отображается
Enabled	Свойство позволяет сделать недоступными (False) все компоненты, которые находятся на панели
Visible	Позволяет скрыть (False) панель

PictureBox

Компонент **PictureBox** обеспечивает отображение иллюстрации (рисунка, фотографии и т. п.). Свойства компонента приведены в табл. 12.16.

Таблица 12.16. Свойства компонента *PictureBox*

Свойство	Описание
Image	Иллюстрация, которая отображается в поле компонента
Size	Размер компонента. Уточняющее свойство <i>Width</i> определяет ширину компонента, <i>Height</i> — высоту
SizeMode	Метод отображения (масштабирования) иллюстрации, если ее размер не соответствует размеру компонента: <ul style="list-style-type: none"> • <i>Normal</i> — масштабирование не выполняется (если размер компонента меньше размера иллюстрации, то отображается только часть иллюстрации); • <i>StretchImage</i> — выполняется масштабирование иллюстрации так, чтобы она занимала всю область компонента (если размер компонента не пропорционален размеру иллюстрации, иллюстрация искажается); • <i>AutoSize</i> — размер компонента автоматически изменяется в соответствии с размером иллюстрации; • <i>CenterImage</i> — центрирование иллюстрации в поле компонента, если размер иллюстрации меньше размера компонента; • <i>Zoom</i> — изменение размера иллюстрации таким образом, чтобы она занимала максимально возможную область компонента и при этом отображалась без искажения (с соблюдением пропорций)
Image.PhysicalDimension	Свойство содержит информацию об истинном размере картинки (иллюстрации), загруженной в поле компонента
Location	Положение компонента (области отображения иллюстрации) на поверхности формы. Уточняющее свойство <i>X</i> определяет расстояние от левой границы области до левой границы формы, уточняющее свойство <i>Y</i> — от верхней границы области до верхней границы клиентской области формы (нижней границы заголовка)
Visible	Признак указывает, отображается ли компонент и, соответственно, иллюстрация на поверхности формы
BorderStyle	Вид границы компонента: <ul style="list-style-type: none"> • <i>None</i> — граница не отображается; • <i>FixedSingle</i> — тонкая; • <i>Fixed3D</i> — объемная

RadioButton

Компонент *RadioButton* представляет собой кнопку (переключатель), состояние которой зависит от состояния других переключателей (компонентов *RadioButton*). Обычно компоненты *RadioButton* объединяют в группу (достигается это путем размещения нескольких компонентов в поле компонента *GroupBox*). В каждый момент времени только один из переключателей группы может находиться в выбранном состоянии (возможна ситуация, когда ни один из переключателей не выбран). Состояния компонентов, принадлежащих разным группам, независимы. Свойства компонента приведены в табл. 12.17.

Таблица 12.17. Свойства компонента `RadioButton`

Свойство	Описание
<code>Text</code>	Текст, который находится справа от переключателя
<code>Checked</code>	Состояние, внешний вид переключателя. Если переключатель выбран, то значение свойства <code>Checked</code> равно <code>True</code> ; если не выбран, то значение свойства <code>Checked</code> равно <code>False</code>
<code>TextAlign</code>	Положение текста в поле отображения. Текст может располагаться в центре поля (<code>MiddleCenter</code>), прижат к левой (<code>MiddleLeft</code>) или правой (<code>MiddleRight</code>) границе. Можно задать и другие способы размещения текста надписи (<code>TopLeft</code> , <code>TopCenter</code> , <code>TopRight</code> , <code>BottomLeft</code> , <code>BottomCenter</code> , <code>BottomRight</code>)
<code>CheckAlign</code>	Положение переключателя в поле компонента. Переключатель может быть прижат к левой верхней границе (<code>TopLeft</code>), прижат к левой границе и находится на равном расстоянии от верхней и нижней границ поля компонента (<code>MiddleLeft</code>). Есть и другие варианты размещения переключателя в поле компонента
<code>Enabled</code>	Свойство позволяет сделать переключатель недоступным (<code>False</code>)
<code>Visible</code>	Свойство позволяет скрыть (<code>False</code>) переключатель
<code>AutoCheck</code>	Свойство определяет, должно ли автоматически изменяться состояние переключателя в результате щелчка на его изображении. По умолчанию значение равно <code>True</code>
<code>FlatStyle</code>	Стиль переключателя. Переключатель может быть обычным (<code>Standard</code>), плоским (<code>Flat</code>) или "всплывающим" (<code>PopUp</code>). Стиль переключателя определяет его поведение при позиционировании указателя мыши на изображении переключателя
<code>Appearance</code>	Определяет вид переключателя. Переключатель может выглядеть обычным образом (<code>Normal</code>) или как кнопка (<code>Button</code>)
<code>Image</code>	Картинка, которая отображается в поле компонента
<code>ImageAlign</code>	Положение картинки в поле компонента. Картина может располагаться в центре (<code>MiddleCenter</code>), быть прижатой к левой (<code>MiddleLeft</code>) или правой (<code>MiddleRight</code>) границе. Можно задать и другие способы размещения картинки на кнопке (<code>TopLeft</code> , <code>TopCenter</code> , <code>TopRight</code> , <code>BottomLeft</code> , <code>BottomCenter</code> , <code>BottomRight</code>)
<code>ImageList</code>	Набор картинок, используемых для обозначения различных состояний переключателя. Представляет собой объект типа <code>ImageList</code> . Чтобы задать значение свойства, в форму приложения нужно добавить компонент <code>ImageList</code>
<code>ImageIndex</code>	Номер (индекс) картинки из набора <code>ImageList</code> , которая отображается в поле компонента

ProgressBar

Компонент `ProgressBar` — это индикатор, который обычно используется для наглядного представления протекания процесса (например, обработки или копирования файлов, загрузки информации из сети и т. п.). Свойства компонента

ProgressBar приведены в табл. 12.18. Следует обратить внимание, что выход значения свойства Value за границы диапазона, заданного свойствами Minimum и Maximum, вызывает исключение.

Таблица 12.18. Свойства компонента ProgressBar

Свойство	Описание
Value	Значение, которое отображается в поле компонента в виде полосы, длина которой пропорциональна значению свойства Value
Minimum	Минимально допустимое значение свойства Value
Maximum	Максимально допустимое значение свойства Value
Step	Приращение (шаг) изменения значения свойства Value при использовании метода PerformStep

SaveFileDialog

Компонент SaveFileDialog представляет собой стандартное диалоговое окно **Сохранить**. Свойства компонента приведены в табл. 12.19.

Таблица 12.19. Свойства компонента SaveFileDialog

Свойство	Описание
Title	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст Сохранить как
FileName	Полное имя файла, которое задал пользователь. В общем случае оно образуется из имени каталога, содержимое которого отображается в диалоговом окне, имени файла, которое пользователь ввел в поле Имя файла , и расширения, заданного значением свойства DefaultExt
DefaultExt	Расширение файла по умолчанию. Если пользователь в поле Имя файла не укажет расширение, то к имени файла будет добавлено расширение (при условии, что значение свойства AddExtension равно True), заданное значением этого свойства
InitialDirectory	Каталог, содержимое которого отображается при появлении диалога на экране
RestoreDirectory	Признак необходимости отображать содержимое каталога, указанного в свойстве InitialDirectory, при каждом появлении окна. Если значение свойства равно False, то при следующем появлении окна отображается содержимое каталога, выбранного пользователем в предыдущий раз
CheckPathExists	Признак необходимости проверки существования каталога, в котором следует сохранить файл. Если указанного каталога нет, то выводится информационное сообщение
CheckFileExists	Признак необходимости проверки существования файла с заданным именем. Если значение свойства равно True и файл с указанным именем уже существует, то появляется окно запроса, в котором пользователь может подтвердить необходимость замены (перезаписи) существующего файла

Таблица 12.19 (окончание)

Свойство	Описание
Filter	Свойство задает описание и фильтр (маску) имени файла. В списке файлов отображаются только те файлы, имена которых соответствуют указанной маске. Описание отображается в поле Тип файла . Например, значение Текст *.txt указывает, что в списке файлов нужно отобразить только файлы с расширением txt
FilterIndex	Если фильтр состоит из нескольких элементов (например, Текст *.txt Все файлы *.*), то значение свойства задает фильтр, который используется в момент появления диалога на экране

TextBox

Компонент TextBox предназначен для ввода данных (строки символов) с клавиатуры. Свойства компонента приведены в табл. 12.20.

Таблица 12.20. Свойства компонента TextBox

Свойство	Описание
Name	Имя компонента. Используется для доступа к компоненту и его свойствам
Text	Текст, который находится в поле редактирования
Location	Положение компонента на поверхности формы
Size	Размер компонента
Font	Шрифт, используемый для отображения текста в поле компонента
ForeColor	Цвет текста, находящегося в поле компонента
BackColor	Цвет фона поля компонента
BorderStyle	Вид рамки (границы) компонента. Граница компонента может быть обычной (Fixed3D), тонкой (FixedSingle) или отсутствовать (None)
TextAlign	Способ выравнивания текста в поле компонента. Текст в поле компонента может быть прижат к левой границе компонента (Left), правой (Right) или находиться по центру (Center)
MaxLength	Максимальное количество символов, которое можно ввести в поле компонента
PasswordChar	Символ, который используется для отображения вводимых пользователем символов (введенная пользователем строка находится в свойстве Text)
Multiline	Разрешает (True) или запрещает (False) ввод нескольких строк текста
ReadOnly	Разрешает (True) или запрещает (False) редактирование отображаемого текста
Dock	Способ привязки положения и размера компонента к размеру формы. По умолчанию привязка отсутствует (None). Если значение свойства равно Top или Bottom, то ширина компонента устанавливается равной ширине формы и компонент прижимается соответственно к верхней или нижней границе формы. Если значение свойства Dock равно Fill, а свойства Multiline — True, то размер компонента устанавливается максимально возможным

Таблица 12.20 (окончание)

Свойство	Описание
Lines	Массив строк, элементы которого содержат текст, находящийся в поле редактирования, если компонент находится в режиме MultiLine. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля
ScrollBars	Задает отображаемые полосы прокрутки: Horizontal — горизонтальная; Vertical — вертикальная; Both — горизонтальная и вертикальная; None — не отображать

ToolTip

Компонент ToolTip — вспомогательный, он используется другими компонентами формы для вывода подсказок при позиционировании указателя мыши на компоненте. Свойства компонента приведены в табл. 12.21.

Таблица 12.21. Свойства компонента ToolTip

Свойство	Описание
Active	Разрешает (True) или запрещает (False) отображение подсказок
AutoPopDelay	Время отображения подсказки
InitialDelay	Время, в течение которого указатель мыши должен быть неподвижным, чтобы появилась подсказка
ReshowDelay	Время задержки отображения подсказки после перемещения указателя мыши с одного компонента на другой

Timer

Компонент Timer генерирует последовательность событий Tick. Свойства компонента приведены в табл. 12.22.

Таблица 12.22. Свойства компонента Timer

Свойство	Описание
Interval	Период генерации события Tick. Задается в миллисекундах
Enabled	Разрешение работы. Разрешает (значение True) или запрещает (значение False) генерацию события Tick

Графика

Графические примитивы

Вычерчивание графических примитивов (линий, прямоугольников, окружностей, дуг, секторов) на графической поверхности выполняют соответствующие методы объекта Graphics (табл. 12.23).

Таблица 12.23. Некоторые методы вычерчивания графических примитивов

Метод	Действие
DrawLine(Pen, x1, y1, x2, y2)	Рисует линию. Параметр Pen определяет цвет, толщину и стиль линии; параметры x1, y1, x2, y2 — координаты точек начала и конца линии
DrawLine(Pen, p1, p2)	Рисует линию. Параметр Pen определяет цвет, толщину и стиль линии; параметры p1 и p2 (структуры Point) — точки начала и конца линии
DrawRectangle(Pen, x, y, w, h)	Рисует контур прямоугольника. Параметр Pen определяет цвет, толщину и стиль границы прямоугольника: параметры x, y — координаты левого верхнего угла; параметры w и h задают размер прямоугольника
DrawRectangle(Pen, rect)	Рисует контур прямоугольника. Параметр Pen определяет цвет, толщину и стиль границы прямоугольника: параметр rect (структура типа Rectangle) — область, граница которой определяет контур прямоугольника
FillRectangle(Brush, x, y, w, h)	Рисует закрашенный прямоугольник. Параметр Brush определяет цвет и стиль закраски прямоугольника; параметры x, y — координаты левого верхнего угла; параметры w и h задают размер прямоугольника
FillRectangle(Brush, rect)	Рисует закрашенный прямоугольник. Параметр Brush определяет цвет и стиль закраски прямоугольника; параметр rect (структура типа Rectangle) — закрашиваемую область
DrawEllipse(Pen, x, y, w, h)	Рисует эллипс (контур). Параметр Pen определяет цвет, толщину и стиль линии эллипса; параметры x, y, w, h — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
DrawEllipse(Pen, rect)	Рисует эллипс (контур). Параметр Pen определяет цвет, толщину и стиль линии эллипса; параметр rect — область, внутри которой рисуется эллипс
FillEllipse(Brush, x, y, w, h)	Рисует закрашенный эллипс. Параметр Brush определяет цвет и стиль закраски внутренней области эллипса; параметры x, y, w, h — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
FillEllipse(Brush, x, y, rect)	Рисует закрашенный эллипс. Параметр Brush определяет цвет и стиль закраски внутренней области эллипса; параметр rect — область, внутри которой вычерчивается эллипс
DrawPolygon(Pen, P)	Рисует контур многоугольника. Параметр Pen определяет цвет, толщину и стиль линии границы многоугольника; параметр P (массив типа Point) — координаты углов многоугольника
FillPolygon(Brush, P)	Рисует закрашенный многоугольник. Параметр Brush определяет цвет и стиль закраски внутренней области многоугольника; параметр P (массив типа Point) — координаты углов многоугольника

Таблица 12.23 (окончание)

Метод	Действие
DrawString(str, Font, Brush, x, y)	Выводит на графическую поверхность строку текста. Параметр <code>Font</code> определяет шрифт; <code>Brush</code> — цвет символов; <code>x</code> и <code>y</code> — точку, от которой будет выведен текст
DrawImage(Image, x, y)	Выводит на графическую поверхность иллюстрацию. Параметр <code>Image</code> определяет иллюстрацию; <code>x</code> и <code>y</code> — координату левого верхнего угла области вывода иллюстрации

Карандаш

Карандаш определяет вид линии — цвет, толщину и стиль. В распоряжении программиста есть набор цветных карандашей (всего их 141), при помощи которых можно рисовать сплошные линии толщиной в *один пиксель* (табл. 12.24).

Таблица 12.24. Некоторые карандаши из стандартного набора

Карандаш	Цвет
Pens.Red	Красный
Pens.Orange	Оранжевый
Pens.Yellow	Желтый
Pens.Green	Зеленый
Pens.LightBlue	Голубой
Pens.Blue	Синий
Pens.Purple	Пурпурный
Pens.Black	Черный
Pens.LightGray	Серый
Pens.White	Белый
Pens.Transparent	Прозрачный

Программист может создать собственный карандаш, объект типа `Pen` и, задав значения свойств (табл. 12.25), определить цвет, толщину и стиль линии, которую он будет рисовать.

Таблица 12.25. Свойства объекта `Pen`

Свойство	Описание
Color	Цвет линии. В качестве значения свойства следует использовать одну из констант <code>Color</code> (например, <code>Color.Red</code>), определенных в пространстве имен <code>System.Drawing</code>

Таблица 12.25 (окончание)

Свойство	Описание
Width	Толщина линии (задается в пикселях)
DashStyle	Вид линии. В качестве значения свойства следует использовать одну из констант DashStyle, определенных в пространстве имен System.Drawing.Drawing2D: <ul style="list-style-type: none"> • DashStyle.Solid — сплошная; • DashStyle.Dash — пунктирная, длинные штрихи; • DashStyle.Dot — пунктирная, короткие штрихи; • DashStyle.DashDot — пунктирная, чередование длинного и короткого штрихов; • DashStyle.DashDotDot — пунктирная, чередование одного длинного и двух коротких штрихов; • DashStyle.Custom — пунктирная линия, вид которой определяет значение свойства DashPattern
DashPattern	Длина штрихов и промежутков пунктирной линии DashStyle.Custom

Кисть

Кисти используются для закраски внутренних областей геометрических фигур. В распоряжении программиста есть четыре типа кистей: стандартные (Brush), штриховые (HatchBrush), градиентные (LinearGradientBrush) и текстурные (TextureBrush).

Стандартная кисть закрашивает область одним цветом (сплошная закраска). В стандартном наборе более 100 кистей, некоторые из них приведены в табл. 12.26.

Таблица 12.26. Некоторые кисти из стандартного набора

Кисть	Цвет
Brushes.Red	Красный
Brushes.Orange	Оранжевый
Brushes.Yellow	Желтый
Brushes.Green	Зеленый
Brushes.LightBlue	Голубой
Brushes.Blue	Синий
Brushes.Purple	Пурпурный
Brushes.Black	Черный
Brushes.LightGray	Серый
Brushes.White	Белый
Brushes.Transparent	Прозрачный

Штриховая кисть (`HatchBrush`) закрашивает область путем штриховки. Область может быть заштрихована горизонтальными, вертикальными или наклонными линиями разного стиля и толщины. В табл. 12.27 перечислены некоторые из возможных стилей штриховки. Полный список стилей штриховки можно найти в справочной системе.

Таблица 12.27. Некоторые стили штриховки областей

Стиль	Штриховка
<code>HatchStyle.LightHorizontal</code>	Редкая горизонтальная
<code>HatchStyle.Horizontal</code>	Средняя горизонтальная
<code>HatchStyle.NarrowHorizontal</code>	Частая горизонтальная
<code>HatchStyle.LightVertical</code>	Редкая вертикальная
<code>HatchStyle.Vertical</code>	Средняя вертикальная
<code>HatchStyle.NarrowVertical</code>	Частая вертикальная
<code>HatchStyle.LargeGrid</code>	Крупная сетка из горизонтальных и вертикальных линий
<code>HatchStyle.SmallGrid</code>	Мелкая сетка из горизонтальных и вертикальных линий
<code>HatchStyle.DottedGrid</code>	Сетка из горизонтальных и вертикальных линий, составленных из точек
<code>HatchStyle.ForwardDiagonal</code>	Диагональная штриховка "вперед"
<code>HatchStyle.BackwardDiagonal</code>	Диагональная штриховка "назад"
<code>HatchStyle.Percent05 — HatchStyle.Percent90</code>	Точки (степень заполнения 5%, 10%, ..., 90%)
<code>HatchStyle.HorizontalBrick</code>	"Кирпичная стена"
<code>HatchStyle.LargeCheckerBoard</code>	"Шахматная доска"
<code>HatchStyle.SolidDiamond</code>	"Бриллиант" ("Шахматная доска", повернутая на 45°)
<code>HatchStyle.Sphere</code>	"Пузырьки"
<code>HatchStyle.ZigZag</code>	"Зигзаг"

Градиентная кисть (`LinearGradientBrush`) представляет собой прямоугольную область, цвет точек которой зависит от расстояния до границы. Обычно градиентные кисти двухцветные, т. е. цвет точек по мере удаления от границы постепенно меняется с одного на другой. Цвет может меняться вдоль горизонтальной или вертикальной границы области. Возможно также изменение цвета вдоль линии, образующей угол с горизонтальной границей.

Текстурная кисть (`TextureBrush`) представляет собой рисунок, который обычно загружается во время работы программы из файла (`bmp`, `jpg` или `gif`) или из ресурса. Закраска текстурной кистью выполняется путем дублирования рисунка внутри области.

Типы данных

Целый тип

Целые типы приведены в табл. 12.28.

Таблица 12.28. Основные целые типы

Тип	Диапазон	Формат
System.SByte	-128 ... 127	8 битов, со знаком
System.Int16	-32768 ... 32767	16 битов, со знаком
System.Int32	-2 147 483 648 ... 2 147 483 647	32 бита, со знаком
System.Int64	- 2^{63} ... 2^{63}	64 бита, со знаком
System.Byte	0 ... 255	8 битов, без знака
System.UInt16	0 ... 65535	16 битов, без знака
System.UInt32	0 ... 4294967295	32 бита, без знака

Вещественный тип

Вещественные типы приведены в табл. 12.29.

Таблица 12.29. Основные вещественные типы

Тип	Диапазон	Значащих цифр	Байтов
System.Single	$-1.5 \cdot 10^{45}$... $3.4 \cdot 10^{38}$	7—8	4
System.Double	$-5.0 \cdot 10^{324}$... $1.7 \cdot 10^{308}$	15—16	8

Символьный и строковый типы

Основной символьный тип — System.Char.

Основной строковый тип — System.String.

Функции

В этом разделе описаны некоторые наиболее часто используемые функции.

Функции преобразования

В табл. 12.30 приведены функции, обеспечивающие преобразование строки символов в число, изображением которого является строка. Если строка не может быть преобразована в число (например, из-за того что содержит недопустимый

символ), то возникает исключение типа `FormatException`. Функции преобразования принадлежат пространству имен `System.Convert`.

Таблица 12.30. Функции преобразования строки в число

Функция	Значение
<code>ToSingle(s)</code> , <code>ToDouble(s)</code>	Дробное типа <code>Single</code> , <code>Double</code>
<code>ToByte(s)</code> , <code>ToInt16(s)</code> , <code>ToInt32(s)</code> , <code>ToInt64(s)</code>	Целое типа <code>Byte</code> , <code>Int16</code> , <code>Int32</code> , <code>Int64</code>
<code>ToUInt16(s)</code> , <code>ToInt32(s)</code> , <code>ToInt64(s)</code>	Целое типа <code>UInt16</code> , <code>UInt32</code> , <code>UInt64</code>

Преобразование числа в строку символов обеспечивает метод (функция) `ToString`. В качестве параметра функции `ToString` можно указать символьную константу (табл. 12.31), которая задает формат строки-результата.

Таблица 12.31. Форматы отображения чисел

Константа	Формат	Пример
<code>c, C</code>	<code>Currency</code> — финансовый (денежный). Используется для представления денежных величин. Обозначение денежной единицы, разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28 р.
<code>e, E</code>	<code>Scientific (exponential)</code> — научный. Используется для представления очень маленьких или очень больших чисел. Разделитель целой и дробной частей числа задается в настройках операционной системы	5,50528+E004
<code>f, F</code>	<code>Fixed</code> — число с фиксированной точкой. Используется для представления дробных чисел. Количество цифр дробной части, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
<code>n, N</code>	<code>Number</code> — числовой. Используется для представления дробных чисел. Количество цифр дробной части, символ-разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
<code>g, G</code>	<code>General</code> — универсальный формат. Похож на <code>Number</code> , но разряды не разделены на группы	55 055,275
<code>r, R</code>	<code>Roundtrip</code> — без округления. В отличие от формата <code>N</code> , этот формат не выполняет округления (количество цифр дробной части зависит от значения числа)	55 055,2775

ФУНКЦИИ МАНИПУЛИРОВАНИЯ СТРОКАМИ

Некоторые функции (методы) манипулирования строками приведены в табл. 12.32.

Инструкция вызова метода в общем виде выглядит так:

`s.Метод(параметры);`

где `s` — строка, над которой надо выполнить операцию.

Пример:

```
String s;
s = "Hello, World!";
int len = s.Length;
s.ToUpper();
```

Таблица 12.32. Функции (методы) манипулирования строками

Функция (метод)	Значение
<code>Length</code>	Длина (количество символов) строки. Строго говоря, <code>Length</code> — это не метод, а свойство объекта <code>String</code>
<code>IndexOf(c)</code>	Положение (номер) символа <code>c</code> в строке <code>s</code> (символы строки нумеруются с нуля). Если указанного символа в строке нет, то значение функции равно минус один
<code>LastIndexOf(c)</code>	Положение (от конца) символа <code>c</code> в строке <code>s</code> (символы строки нумеруются с нуля). Если указанного символа в строке нет, то значение функции равно минус один
<code>IndexOf(st)</code>	Положение подстроки <code>st</code> в строке <code>s</code> (символы строки <code>s</code> нумеруются с нуля). Если указанной подстроки в строке нет, то значение функции равно минус один
<code>Trim()</code>	Строка, полученная из строки <code>s</code> путем "отбрасывания" пробелов, находящихся в начале и в конце строки
<code>Substring(n)</code>	Подстрока, выделенная из строки <code>s</code> , начиная с символа <code>s</code> с номером <code>n</code> (символы строки <code>s</code> нумеруются с нуля). Если значение <code>n</code> больше, чем количество символов в строке, то возникает исключение <code>ArgumentOutOfRangeException</code>
<code>Substring(n, k)</code>	Подстрока длиной <code>k</code> символов, выделенная из строки <code>s</code> , начиная с символа <code>s</code> с номером <code>n</code> (символы строки <code>s</code> нумеруются с нуля). Если значение <code>n</code> больше, чем количество символов в строке, или если <code>k</code> больше, чем <code>len - n</code> (где <code>len</code> — длина строки <code>s</code>), то возникает исключение <code>ArgumentOutOfRangeException</code>
<code>Insert(pos, st)</code>	Строка, полученная путем вставки в строку <code>s</code> строки <code>st</code> . Параметр <code>pos</code> задает номер символа строки <code>s</code> , после которого вставляется строка <code>st</code>
<code>Remove(pos, n)</code>	Строка, полученная путем удаления из строки <code>s</code> цепочки символов (подстроки). Параметр <code>pos</code> задает положение подстроки, параметр <code>n</code> — количество символов, которое нужно удалить. Если значение <code>pos</code> больше, чем количество символов в строке, или если <code>n</code> больше, чем <code>len - pos</code> (где <code>len</code> — длина строки <code>s</code>), то возникает исключение <code>ArgumentOutOfRangeException</code>
<code>Replace(old, new)</code>	Строка, полученная из строки <code>s</code> путем замены всех символов <code>old</code> на символ <code>new</code>
<code>ToUpper()</code>	Строка, полученная из строки <code>s</code> путем замены строчных символов на прописные

Таблица 12.32 (окончание)

Функция (метод)	Значение
ToLower()	Строка, полученная из строки s путем замены прописных символов на строчные
Split(sep)	Массив строк, полученный разбиением строки s на подстроки. Параметр sep (массив типа Char) задает символы, которые используются методом Split для обнаружения границ подстрок

ФУНКЦИИ МАНИПУЛИРОВАНИЯ ДАТАМИ И ВРЕМЕНЕМ

Некоторые функции (методы) манипулирования датами приведены в табл. 12.33. Инструкция вызова метода в общем виде выглядит так:

d.Метод(параметры);

где d — объект DateTime, над которым надо выполнить операцию.

Пример:

```
DateTime d1;
String s;
d1 = DateTime.Today;
s = d1.ToString();
```

Таблица 12.33. Функции манипулирования датами и временем

Функция (метод)	Значение
DateTime.Now	Структура типа DateTime. Текущие дата и время
DateTime.Today	Структура типа DateTime. Текущая дата
ToString()	Строка вида dd.mm.yyyy mm:ss (например, 05.06.2010 10:00)
ToString(f)	Строка — дата и/или время. Вид строки определяет параметр f. Полному формату даты и времени соответствует строка dd.MM.yyyy hh:mm:ss
Day	День (номер дня месяца)
Month	Номер месяца (1 — январь, 2 — февраль и т. д.)
Year	Год
Hour	Час
Minute	Минуты
DayOfYear	Номер дня года (отсчет от 1 января)
DayOfWeek	День недели
ToLongDateString()	"Длинная" дата. Например: 5 июня 2010
ToShortDateString()	"Короткая" дата. Например: 05.06.2010
ToLongTimeString()	Время в формате hh:mm:ss

Таблица 12.33 (окончание)

Функция (метод)	Значение
ToShortTimeString()	Время в формате hh:mm
AddDays (n)	Дата, отстоящая от d на n дней. Положительное n "сдвигает" дату вперед, отрицательное — назад
AddMonths (n)	Дата, отстоящая от d на n месяцев. Положительное n "сдвигает" дату вперед, отрицательное — назад
AddYears (n)	Дата, отстоящая от d на n лет. Положительное n "сдвигает" дату вперед, отрицательное — назад
AddHours (n)	Дата (время), отстоящая от d на n часов. Положительное n "сдвигает" дату вперед, отрицательное — назад
Minutes (τ)	Дата (время), отстоящая от d на n минут. Положительное n "сдвигает" дату вперед, отрицательное — назад

Функции манипулирования каталогами и файлами

Функции (методы) манипулирования каталогами и файлами (табл. 12.34) принадлежат пространству имен System.IO. При выполнении операций с каталогами и файлами возможны исключения.

Обозначения:

- ◆ di — объект DirectoryInfo;
- ◆ fi — объект FileInfo;
- ◆ sr — объект StreamReader;
- ◆ sw — объект StreamWriter.

Таблица 12.34. Функции (методы) манипулирования каталогами и файлами

Функция (метод)	Результат (значение)
DirectoryInfo (Path)	Создает объект типа DirectoryInfo, соответствующий каталогу, заданному параметром Path
di.GetFiles (fn)	Формирует список файлов каталога — массив объектов типа FileInfo. Каждый элемент массива соответствует файлу каталога, заданного объектом di (тип DirectoryInfo). Параметр fn задает критерий отбора файлов
di.Exists	Проверяет, существует ли в системе каталог. Если каталог существует, то значение функции равно True, в противном случае — False
di.Create (dn)	Создает каталог. Если путь к новому каталогу указан неправильно, возникает исключение
fi.CopyTo (Path)	Копирует файл, заданный объектом fi (тип FileInfo), в каталог и под именем, заданным параметром Path. Значение метода — объект типа FileInfo, соответствующий файлу, созданному в результате копирования

Таблица 12.34 (продолжение)

Функция (метод)	Результат (значение)
fi.MoveTo(Path)	Перемещает файл, заданный объектом fi (тип FileInfo), в каталог и под именем, заданным параметром Path
fi.Delete()	Уничтожает файл, соответствующий объекту fi (тип FileInfo)
StreamReader(fn)	Создает и открывает для чтения поток, соответствующий файлу, заданному параметром fn. Значение метода — объект типа StreamReader. Поток открывается для чтения в формате UTF-8
StreamReader (fn, encd)	Создает и открывает для чтения поток, соответствующий файлу, заданному параметром fn. Значение метода — объект типа StreamReader. Поток открывается для чтения в кодировке, заданной параметром encd (объект типа System.Text.Encoding). Для чтения текста в кодировке Windows 1251 параметр encd необходимо инициализировать значением System.Text.Encoding.GetEncoding(1251)
sr.Read()	Читает символ из потока sr (объект типа StreamReader). Значение метода — код символа
sr.Read(buf, p, n)	Читает из потока sr (объект типа StreamReader) символы и записывает их в массив символов buf (тип Char). Параметр p задает номер элемента массива, в который будет помещен первый прочитанный символ, параметр n — количество символов, которое нужно прочитать
sr.ReadToEnd()	Читает весь текст из потока sr (объект типа StreamReader). Значение метода — прочитанный текст
sr.ReadLine()	Читает строку из потока sr (объект типа StreamReader). Значение метода — прочитанная строка
StreamWriter(fn)	Создает и открывает для записи поток, соответствующий файлу, заданному параметром fn. Значение метода — объект типа StreamWriter. Поток открывается для записи в формате (кодировке) UTF-8
StreamWriter(fn, a, encd)	Создает и открывает для записи поток, соответствующий файлу, заданному параметром fn. Значение метода — объект типа StreamWriter. Поток открывается для записи в кодировке, заданной параметром encd (объект типа System.Text.Encoding). Для записи текста в кодировке Windows 1251 параметр encd необходимо инициализировать значением System.Text.Encoding.GetEncoding(1251). Параметр a задает режим записи: True — добавление в файл, False — перезапись
sw.WriteLine(v)	Записывает в поток sw (объект типа StreamWriter) строку символов, соответствующую значению v. В качестве v можно использовать выражение строкового или числового типа
sw.WriteLine(v)	Записывает в поток sw (объект типа StreamWriter) строку символов, соответствующую значению v, и символ "новая строка". В качестве v можно использовать выражение строкового или числового типа

Таблица 12.34 (окончание)

Функция (метод)	Результат (значение)
sw.WriteLine	Записывает в поток sw (объект типа StreamWriter) символ "новая строка"
s.Close()	Закрывает поток s

Функции обработки имен файлов

Функции обработки имен файлов (табл. 12.35) принадлежат пространству имен System.IO.Path.

Таблица 12.35. Функции обработки имен файлов

Функция (метод)	Результат (значение)
GetDirectoryName(s)	Выделяет из полного имени файла имя каталога
GetExtension(s)	Выделяет из полного имени файла расширение файла
GetFileName(s)	Выделяет из полного имени файла имя файла
GetFileNameWithoutExtension(s)	Выделяет из имени файла имя файла без расширения
GetFullPath(s)	Возвращает полное имя файла
GetPathRoot(s)	Возвращает имя корневого каталога
Combine(s1, s2)	Объединяет две строки, например путь и имя файла, в путь

Математические функции

Некоторые математические функции, принадлежащие пространству имен Math, приведены в табл. 12.36.

Таблица 12.36. Математические функции

Функция (метод)	Значение
Abs(n)	Абсолютное значение n
Sign(n)	1 (если n — положительное), -1 (если n — отрицательное)
Round(n, m)	Дробное, полученное путем округления n по известным правилам. Параметр m задает количество цифр дробной части
Ceiling(n)	Дробное, полученное путем округления n "с избытком"
Floor(n)	Дробное, полученное путем округления n "с недостатком" (отбрасыванием дробной части)

Таблица 12.36 (окончание)

Функция (метод)	Значение
Log (n)	Натуральный логарифм n (логарифм n по основанию E, где E — постоянная, значение которой равно 2.7182818284590452354)
Log (n, m)	Логарифм n по основанию m
Log10 (n)	Десятичный логарифм n
Exp (n)	Экспонента n (число "E" в степени n)
Cos (α)	Косинус угла α , заданного в радианах
Sin (α)	Синус угла α , заданного в радианах
Tan (α)	Тангенс угла α , заданного в радианах
ASin (n)	Арксинус n — угол (в радианах), синус которого равен n
ACos (n)	Арккосинус n — угол (в радианах), косинус которого равен n
ATan (n)	Арктангенс n — угол (в радианах), тангенс которого равен n
Sqrt (n)	Квадратный корень из n
r.Next (hb)	Случайное число из диапазона 0 ... (hb-1), где r — объект типа System.Random

Величина угла тригонометрических функций должна быть задана в радианах. Чтобы преобразовать величину угла из градусов в радианы, нужно значение, выраженное в градусах, умножить на $\pi/180$. Числовую константу 3.1415926 (значение числа π) можно заменить именованной константой PI.

События

В табл. 12.37 приведено краткое описание некоторых событий.

Таблица 12.37. События

Событие	Происходит
Click	При щелчке кнопкой мыши
Closed	Сразу за событием Closing
Closing	Как результат щелчка на системной кнопке Закрыть
DblClick	При двойном щелчке кнопкой мыши
Enter	При получении элементом управления фокуса
KeyDown	При нажатии клавиши. Сразу за событием KeyDown возникает событие KeyPress. Если нажатая клавиша удерживается, то событие KeyDown возникает еще раз. Таким образом, пара событий KeyDown и KeyPress генерируется до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие KeyUp)

Таблица 12.37 (окончание)

Событие	Происходит
KeyPress	При нажатии клавиши. Событие KeyPress возникает сразу после события KeyDown
KeyUp	При отпускании нажатой клавиши
Leave	При потере элементом управления фокуса
Load	В момент загрузки формы. Используется для инициализации программы
MouseDown	При нажатии кнопки мыши
MouseMove	При перемещении мыши
MouseUp	При отпускании кнопки мыши
Paint	При появлении окна (элемента управления) на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном, и в других случаях. Только процедура обработки события Paint имеет доступ к свойству Graphics, методы которого обеспечивают отображение графики
Resize	При изменении размера окна. Если размер формы увеличивается, то сначала происходит событие Paint, затем — Resize. Если размер формы уменьшается, то происходит только событие Resize

Исключения

В табл. 12.38 перечислены наиболее часто возникающие исключения и указаны причины, которые могут привести к их возникновению.

Таблица 12.38. Исключения

Исключение	Возникает
FormatException — ошибка преобразования	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому типу. Наиболее часто возникает при преобразовании строки символов в число
IndexOutOfRangeException — выход значения индекса за допустимые границы	При обращении к несуществующему элементу массива
ArgumentOutOfRangeException — выход значения аргумента за допустимые границы	При обращении к несуществующему элементу данных, например, при выполнении операций со строками
OverflowException — переполнение	Если результат выполнения операции выходит за границы допустимого диапазона, а также при выполнении операции деления, если делитель равен нулю

Таблица 12.38 (окончание)

Исключение	Возникает
FileNotFoundException — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частая причина — отсутствие требуемого файла (ошибка в имени файла или обращение к несуществующему или недоступному устройству)
DirectoryNotFoundException — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частая причина — отсутствие требуемого каталога (ошибка в имени каталога или обращение к несуществующему или недоступному устройству)

ПРИЛОЖЕНИЕ

Описание компакт-диска

Прилагаемый компакт-диск содержит проекты Microsoft Visual C# 2010, приведенные в книге. Каждый проект находится в отдельном каталоге.

Для активной работы, чтобы иметь возможность вносить изменения в программы, скопируйте каталоги проектов на жесткий диск компьютера, в папку проектов Microsoft Visual Studio (по умолчанию это C:\Users\User\Documents\Visual Studio 2010, где *User* — имя пользователя в системе).

Чтобы увидеть, как работает приложение, загрузите проект в Microsoft Visual C#, откомпилируйте его и затем запустите. Следует обратить внимание, некоторые программы, например работы с базами данных, требуют настройки соединения с базой данных с учетом реального ее размещения.

Предметный указатель

A

ADODataset 196

B

Bitmap 172

D

DrawString 167

E

Event 35

F

Font 168

G

Graphics 141

H

HTML Help 263

HTML-теги 265

I

Internet 319

L

LINQ:

◊ запись в XML-файл 248

◊ обработка массива 241

◊ записей 243

◊ отображение XML-документа 245

◊ поиск в массиве 240

◊ создание XML-файла 248

◊ сортировка массива 241

◊ чтение из XML-файла 248

M

Microsoft HTML Help Workshop 263

P

Point 153

Q

Query-оператор 237

Q-оператор 237

R

Rectangle 153

S

ShowModal 318

SQL-запрос 204

SQL-команда SELECT 204

T

Try ... Except 50

Б

База данных, просмотр 200
 Битовый образ 172
 Браузер 319

В

Внешняя программа, запуск 318
 Время:
 ◇ минуты 352
 ◇ текущее 352
 ◇ формат отображения 352
 ◇ час 352

Г

Графические примитивы:
 ◇ дуга 161
 ◇ линия 154
 ◇ ломаная линия 158
 ◇ многоугольник 160
 ◇ прямоугольник 158
 ◇ эллипс 160
 Графический примитив 152

Д

Дата 352
 ◇ год 352
 ◇ день недели 352
 ◇ месяц 352
 ◇ текущая 352
 ◇ формат отображения 352
 Диалог "О программе" 317
 Дуга 161

З

Запуск программы 49

И

Инструкция try 255
 Исключение 49, 253
 ◇ FileNotFoundException 358
 ◇ FormatException 51, 357
 ◇ OverflowException 51, 357
 ◇ обработка 50, 255

К

Карандаш 144, 346
 ◇ программиста 145
 ◇ системный набор 145
 ◇ стандартный набор 144
 ◇ цвет 145
 Кисть:
 ◇ градиентная 348
 ◇ стандартная 147, 347
 ◇ текстурная 149, 348
 ◇ штриховая 148, 348
 Командная строка 292
 Компиляция 46
 Компонент 26, 61
 ◇ ADODataSet 196
 ◇ Button 33, 68, 330
 ◇ CheckBox 72, 333
 ◇ CheckedListBox 106, 334
 ◇ ComboBox 83, 331
 ◇ ContextMenuStrip 332
 ◇ DataGridView 200
 ◇ GroupBox 79, 334
 ◇ ImageList 101, 335
 ◇ Label 61, 335
 ◇ ListBox 94, 336
 ◇ ListView 98, 245
 ◇ MainMenu 309
 ◇ MenuStrip 125, 337
 ◇ NotifyIcon 117, 337
 ◇ NumericUpDown 111, 338
 ◇ OpenFileDialog 128, 338
 ◇ Panel 105, 339
 ◇ PictureBox 88, 339
 ◇ ProgressBar 341
 ◇ RadioButton 76, 340
 ◇ SaveFileDialog 130, 342
 ◇ StatusStrip 115
 ◇ TextBox 27, 64, 343
 ◇ Timer 108, 344
 ◇ ToolStrip 123
 ◇ ToolTip 103, 344

Л

Линия:
 ◇ стиль 145, 347
 ◇ толщина 144, 145, 346, 347
 ◇ цвет 144, 145, 346
 Лямбда-выражение 237

M

- Меню 125
 Метод:
 ◊ DrawArc 161
 ◊ DrawEllipse 160
 ◊ DrawImage 172, 346
 ◊ DrawLine 154, 345
 ◊ DrawLines 158
 ◊ DrawPolygon 160
 ◊ DrawRectangle 345
 ◊ DrawString 167, 346
 ◊ FillEllipse 161, 345
 ◊ FillPolygon 345
 ◊ FillRectangle 159, 345
 Многоугольник 160
 Модальный диалог 318
 Модуль:
 ◊ главный 40
 ◊ формы 42

O

- Объект, свойства 16
 Отладка 253
 ◊ по шагам 258
 Отладчик 258
 Ошибка 253
 ◊ алгоритмическая 255
 ◊ времени выполнения 253
 ◊ синтаксическая 253

П

- Панель инструментов 123
 Перегрузка 153
 Переменная, контроль значения 260
 Проект, структура 40
 Пространство имен 39
 Процедура обработки события 36
 ◊ параметры 37
 Прямоугольник 158

C

- Событие 35
 ◊ Click 35, 356
 ◊ Create 35
 ◊ DblClick 35, 356
 ◊ Enter 35, 356

- ◊ Exit 35
 ◊ KeyDown 35, 356
 ◊ KeyPress 35, 357
 ◊ KeyUp 35, 357
 ◊ MouseDown 35, 357
 ◊MouseMove 35, 357
 ◊ MouseUp 35, 357
 ◊ Paint 35, 357
 ◊ обработка 36
 Стока:
 ◊ вставка подстроки 351
 ◊ длина 351
 ◊ замена подстроки 351
 ◊ поиск подстроки 351
 ◊ поиск символа 351
 ◊ удаление подстроки 351
 Стока состояния 115

T

- Тестирование 253
 Тип:
 ◊ Byte 349
 ◊ Double 349
 ◊ Int16 349
 ◊ Int32 349
 ◊ Int64 349
 ◊ SByte 349
 ◊ Single 349
 ◊ UInt16 349
 ◊ UInt32 349
 Точка останова 258
 Трассировка 258

Ф

- Форма 22
 Формат:
 ◊ Currency 40, 350
 ◊ Fixed 40, 350
 ◊ General 40, 350
 ◊ Number 40, 350
 ◊ Roundtrip 40, 350
 ◊ Scientific 40, 350
 ◊ без округления 40, 350
 ◊ научный 40, 350
 ◊ универсальный 40, 350
 ◊ фиксированная точка 40, 350
 ◊ финансовый 40, 350

Функции:

- ◊ манипулирования датами 352
- ◊ математические 355
- ◊ обработки имен файлов 355
- ◊ файловая система 353

Функция:

- ◊ ToByte 39, 350
- ◊ ToDouble 39, 350
- ◊ToInt16 39, 350
- ◊ToInt32 39, 350
- ◊ToInt64 39, 350
- ◊ ToSingle 39, 350
- ◊ ToUInt16 39, 350
- ◊ ToUInt32 39, 350
- ◊ ToUInt64 39, 350

Ц**Цвет:**

- ◊ закраски области 347
- ◊ карандаша 145
- ◊ линии 144, 145, 346

Ш

- Шрифт, создание 168
Штриховка 148, 348

Э

- Эллипс 160