# C#

# СОВЕТЫ ПРОГРАММИСТАМ

+ CD

- Использование стандартных алгоритмов
- Использование особенностей интегрированной среды разработки IDE
- Работа с формами, элементами управления
- Использование технологий WMI, WSH
- Вызов системных функций Windows API

АЛЕКСАНДР КЛИМОВ

**Александр Климов**

# C#

# СОВЕТЫ ПРОГРАММИСТАМ

Санкт-Петербург
«БХВ-Петербург»
2008

УДК 681.3.068+800.92C#
ББК 32.973.26-018.1
К49

**Климов А. П.**

К49 C#. Советы программистам. — СПб.: БХВ-Петербург, 2008. — 544 с.: ил. + CD-ROM

ISBN 978-5-9775-0174-3

Книга представляет собой сборник советов, алгоритмов и готовых примеров программ на языке C# в среде MS Visual Studio 2005/2008 из различных областей: работа с формами и элементами управления, папками и файлами, мышью и клавиатурой, мультимедиа и графикой, использование технологий WMI и WSH, взаимодействие с MS Office и другими приложениями, работа в локальной сети и Интернете, особенности использования функций Windows API и др.

На компакт-диске размещены примеры из книги, а также демонстрационная версия справочника по функциям Windows API для .NET Framework и сообщениям Windows для Visual Basic .NET и C#.

*Для программистов*

УДК 681.3.068+800.92C#
ББК 32.973.26-018.1

**Группа подготовки издания:**

| | |
|---|---|
| Главный редактор | *Екатерина Кондукова* |
| Зам. главного редактора | *Евгений Рыбаков* |
| Зав. редакцией | *Григорий Добин* |
| Редактор | *Анна Кузьмина* |
| Компьютерная верстка | *Натальи Смирновой* |
| Корректор | *Наталия Першакова* |
| Дизайн обложки | *Елены Беляевой* |
| Зав. производством | *Николай Тверских* |

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 26.02.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 43,86.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.002108.02.07 от 28.02.2007 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

ISBN 978-5-9775-0174-3

# Оглавление

# Вступление

Когда читатель берет в руки новую книгу, он пытается понять, действительно ли она ему нужна. Надеюсь, что эта книга оправдает ваши ожидания. В ней собраны самые разные советы, которые я отбирал для себя, чтобы впоследствии использовать их в своей практике. Нет никакого смысла изобретать заново велосипед, когда уже существует готовое решение, которое сэкономит время и силы. Я попытался выстроить систему советов от простого к сложному, хотя, конечно, это деление весьма условно. Надо сказать, что подборки советов по наиболее стандартным вопросам существуют для всех языков программирования. Достаточно набрать в поисковой системе ключевые слова "faq", "tips", "tricks" или их русские аналоги "чаво", "советы", "трюки", и вы увидите, как много сайтов имеют специальные страницы, на которых собраны самые популярные вопросы и ответы программистов. В конце книги вы найдете главу, в которой перечислены наиболее популярные интернет-сайты, посвященные советам по языку C# и .NET Framework. Рекомендуется регулярно посещать эти страницы, так как советы постоянно пополняются и обновляются вслед за выходом новых версий самого языка C# и платформы .NET Framework.

Автор не ставил перед собой цель охватить весь спектр вопросов, которые возникают у программистов, иначе книга писалась бы до бесконечности. Единственное отличие книжных советов от советов, выложенных на сайтах, — более подробное объяснение или комментарии. Бывает так, что ответ на какой-нибудь вопрос слишком лаконичен, и читателю (особенно новичку) не вполне ясно, когда возникает подобная проблема и почему нужно обратить внимание на ту или иную деталь. Все советы имеют практическую ценность, так как программисты не задают вопросы из праздного любопытства. Кто-то проводит бессонные ночи, пытаясь решить поставленную задачу, и понимает, что его знаний не хватает для правильного ответа. Тогда он принимает решение обратиться на форумы за помощью. И находятся добрые люди, которые уже сталкивались с этой проблемой, и они отвечают или дают рекомендации, которые направят программиста в нужное русло в поисках решения. Практически на любом форуме есть такие бескорыстные помощники, которые много знают в своей области и всегда готовы поделиться своими знаниями. Я очень уважаю таких людей, которые находят время, чтобы исследовать чужую проблему, поковыряться в коде и посоветовать свое решение. Огромное им спасибо!

Книгу не обязательно начинать читать с первой страницы. Используйте книгу как справочный материал. Просмотрите оглавление книги и, если увидите тему, которая вас особенно интересует, начните прямо с нее. Может, вы откроете для себя что-то новое, и прочитанная глава даст вам толчок в развитии давно заброшенного проекта. А потом обязательно прочтите остальные главы, потому что я уверен, что и другие полезные советы тоже пригодятся вам в работе.

## Для кого предназначена книга

Эта книга будет интересна программистам, пишущим свои программы на языке C#. Книга не является самоучителем. В магазинах имеется достаточно большой выбор книг для начинающих, в которых излагаются основы программирования. Предполагается, что читатель уже знаком с основами языка C# и .NET Framework и хотел бы расширить свои познания. У каждого программиста свой уровень подготовки, и трудно угодить всем. Поэтому в книге были собраны советы разной степени сложности.

## Благодарности

В книге приведены сотни советов от самых разных людей. Авторство многих советов уже невозможно установить, но, тем не менее, огромная благодарность их авторам от людей, которые делают первые шаги в программировании. Я тоже говорю огромное спасибо этим авторам, благодаря которым и я изучал тонкости языка, будь то C#, Visual Basic, C++, Delphi, JavaScript и т. д. Однако в книге присутствуют и советы, чье авторство известно. И я хочу отдельно выразить свою признательность этим людям.

## Требования

Большинство примеров в книге, как это видно из названия, написаны на языке программирования C# с использованием среды разработки Visual Studio. Когда я только приступал к работе над книгой, то писал примеры на Visual Studio 2005. Уже во время работы над книгой появилась бета-версия Visual Studio 2008, и я начал проверять написанные примеры заново. А когда я заканчивал написание книги, то вышла и финальная версия Visual Studio 2008.

Чтобы не отставать от времени, я переделал все проекты под новую версию среды разработки, воспользовавшись мастером обновлений. Таким образом, представленные на прилагаемом к книге компакт-диске примеры можно использовать как в Visual Studio 2008, так и в Visual Studio 2005, а некоторые примеры подойдут и для Visual Studio .NET 2003.

На самом деле в приводимых примерах важна сама идея. Поэтому опытные программисты могут воспроизвести код из примеров в среде разработки, которой они привыкли пользоваться. А те, кто имеет опыт написания программ на разных языках, могут переложить код из книги на другой язык. Я сам иногда дублировал код, написанный на C#, на Visual Basic .NET или, наоборот, из Visual Basic .NET переводил код на язык C#. Предполагается, что все примеры будут использоваться на платформе .NET Framework 2.0, хотя на самом деле большинство советов прекрасно подходит для .NET Framework 1.1. В некоторых случаях я буду намеренно обращать внимание на новинки, которые появились в новой версии .NET Framework 2.0.

Несомненно, новая версия среды разработки Visual Studio 2008 привнесет с собой новые советы и трюки. Уже сейчас многие аналитики говорят о новой революции в программировании. Как вы уже догадались, речь идет об особом языке запросов LINQ, который встроен в Visual Studio 2008. В конце книги я приведу несколько примеров использования LINQ, хотя эта тема заслуживает отдельной книги. Также надо отметить тот факт, что в 2007 году вышла в свет новая операционная система Windows Vista. Надо признать, что отношение к этой системе в России, да и во всем мире, сложилось неоднозначное. Но, тем не менее, нам все равно придется переходить на эту систему. Поэтому в конце книги я приведу несколько примеров, которые используют новые возможности Windows Vista.

## Чего вы не найдете в этой книге

Книга — не резиновая игрушка, которую можно растягивать до невероятных размеров. Поэтому считаю своим долгом предупредить читателя, чего вы не найдете в этой книге. Есть темы, которые требуют отдельного разговора и которые интересны более специализированной группе разработчиков. Например, в этой книге вы не найдете советов, связанных с ASP.NET 2.0 или Microsoft SQL Server 2005. Безусловно, разработка веб-приложений или баз данных является важной задачей для многих программистов, но поищите советы в других книгах. Кроме того, в книге не будет советов, относящихся к платформе .NET Framework 3.0. Возможно, вы в курсе, что название .NET

Framework 3.0 является всего лишь маркетинговым ходом. На самом деле, платформа .NET Framework 3.0 является комбинацией .NET Framework 2.0 и новых библиотек Workflow. Эти библиотеки — тоже тема для отдельной книги.

## Исходные коды

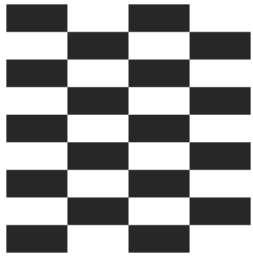
Я всячески рекомендую писать код своими руками, а не копировать из готовых источников. Но если код слишком велик, а идея совета понятна, то возникает соблазн скопировать нужный кусок кода и вставить в свой проект. Поэтому все представленные советы можно найти на компакт-диске, который в обязательном порядке должен прилагаться к книге. Все примеры распределены по папкам, которые имеют названия, соответствующие номерам глав. Внутри каждой из этих папок находятся дополнительные папки, в которых уже содержатся файлы проектов. Если вы пользуетесь средой разработки Visual Studio 2008, то вам достаточно отыскать в папке файл с расширением SLN и запустить его.

Кроме того, на диске вы найдете электронный справочник по функциям Windows API и сообщениям Windows для Visual Basic .NET и C#. На самом деле справочник является платным, и я стараюсь ежемесячно обновлять его. Но вы получите один выпуск справочника совершенно бесплатно! Это небольшой подарок от меня. Если справочник вам понравится, то вы можете приобрести его за небольшую плату и получать обновления.

## Обратная связь

Если вы считаете, что я незаслуженно обошел в книге какие-то темы, сообщите мне об этом. Я постараюсь учесть все пожелания читателей и исправиться в следующей книге. Кроме того, на сайте **http://netsources.narod.ru** будут выкладываться интересные исходные коды, которые могут заинтересовать вас. Если вы захотите поделиться своим интересным проектом, то можете прислать его на мой электронный адрес **rusproject@mail.ru**. Жду ваших писем!

## Глава 1



# Общее

Прежде чем мы перейдем к трюкам, нужно сказать несколько общих слов о простых вещах, которые помогут нам настроиться на нужный лад. Опытные программисты могут пропустить эту главу.

## Часто задаваемые вопросы

Многие новички, которые только начинают осваивать новый для себя язык C#, ищут ответы на довольно простые вопросы. Мы рассмотрим некоторые из них.

### Можно ли запустить программу, написанную на C#, без .NET Framework?

Нет, нельзя. Чтобы ваша программа, написанная на C#, заработала на компьютере пользователя, необходимо установить специальные библиотеки. Скачать необходимые компоненты можно бесплатно с сайта Microsoft. В операционную систему Windows Vista платформа .NET Framework 2.0 уже встроена. Также она поставляется вместе с пакетом обновлений Service Pack 2 для Windows XP.

### В каком редакторе писать программы?

На самом деле программы можно писать даже в стандартном Блокноте, который имеется в любой версии Windows. Очень многие профессиональные программисты так и поступают в отдельных случаях. При таком подходе создается полностью чистый код без добавления различных дополнительных

строк кода, который вставляют визуальные среды разработки. Я думаю, каждый должен хотя бы раз написать программу таким способом, тем более что здесь нет ничего сложного. Для того чтобы писать программы таким способом, вам необходимо скачать .NET Framework SDK, в состав которого входит компилятор csc.exe. При помощи данного компилятора и создается полноценное приложение, написанное на чистом C#. Запустите программу Блокнот и напишите строки, приведенные в листинге 1.1.

**Листинг 1.1. Простая программа, написанная в Блокноте**

```
// Консольная программа на C#,
// написанная в Блокноте
class NotepadProgram
{
    public static void Main()
    {
        System.Console.WriteLine("C#. Трюки для программиста");
        System.Console.ReadLine();
    }
}
```

Сохраните текст программы, приведенный в листинге 1.1, в файле под именем notepad.cs и запустите в командной строке команду

```
csc.exe notepad.csc
```

В результате у вас получится файл notepad.exe. Более подробно процесс создания программ при помощи блокнота описывать мы не будем, желающие сами найдут дополнительную информацию по этому вопросу. Кроме консольных приложений, можно создавать и обычные Windows-приложения. В этом случае в командной строке нужно указывать дополнительные параметры.

Но будем реалистами, подавляющее число программистов не используют отдельный текстовый редактор и компилятор для создания программ. Поэтому не будем надолго задерживаться на этой теме. Можно только добавить, что существуют более удобные оболочки для написания кода, чем Блокнот, имеющие подсветку синтаксиса, выбор различных параметров компилятора и т. д. Но все же большинство программистов используют пакет Visual Studio 2008 или ее усеченный вариант Visual C# 2008. Хочется отметить прекрасный подарок от Microsoft — выпуск бесплатных Express-версий Visual Studio 2008, с помощью которых можно писать полноценные приложения практически любой сложности.

Далее я приведу несколько простых советов и правил, которые помогут новичкам быстрее освоить новый язык программирования.

## Использование в качестве переменных русских символов

В отличие, скажем, от Visual Basic 6.0, в C# используются символы Unicode при написании кода. Поэтому вы можете выбирать в качестве имен переменных, например, буквы греческого алфавита, иврит или русские буквы. Таким образом, можно изменить программу так, что она будет похожа на программу, написанную на встроенном языке 1С, как в листинге 1.2.

**Листинг 1.2. Использование русских символов в коде**

```
// Русские псевдонимы
using целое = System.Int16;
using дробное = System.Single;


целое переменная1 = 7;
дробное переменная2 = 3.14F;
```

## Псевдонимы

Обратите внимание на оператор `using`. Как правило, этот оператор используют для указания пространства имен, что позволяет сократить написание кода. Но оператор `using` можно применять не только для указания пространств имен, но и для сокращенной записи классов, то есть использовать его для описания псевдонимов. Запомните, что псевдонимы записываются вне описания класса. В приведенном выше примере мы используем псевдоним `целое` для целочисленного типа `System.Int16`. Чтобы закрепить материал с псевдонимами, приведу еще один пример (листинг 1.3).

**Листинг 1.3. Использование псевдонимов**

```
using System;
using строка = System.String;  // псевдоним строка для строкового типа
                               // System.String
```

```
class MainClass
{

     static void Main()
    {
        строка ИмяКота = "Рыжик";
        Console.WriteLine("Моего кота зовут " + ИмяКота);
    }
}
```

## Копирующий строковой литерал

В C# можно использовать специальный копирующий строковой литерал. Копирующий строковой литерал начинается с символа `@`, за которым следует строка в кавычках. Во многих случаях это очень удобно. Например, подобный литерал часто используется при указании полного имени файла. Без использования этого литерала пришлось бы дублировать обратные слэши, используемые в пути файла:

```
string filename = "c:\\windows\\temp\\samples\\test.txt";
```

Дублировать слэши приходится из-за того, что компилятор принимает их за начало escape-последовательности. Но если вы поставите знак `@`, а содержимое строки заключите в кавычки, то сама строка в этом случае принимается без модификации:

```
string filename = @"c:\windows\temp\samples\test.txt";
// так проще и удобнее
```

## Символ @ перед идентификатором

Символ `@` также можно использовать перед идентификатором. Такая позиция символа `@` позволяет использовать имя, даже если оно является ключевым словом. На самом деле эта возможность вряд ли пригодится в вашей практике. Но, возможно, вам придется подключать какую-нибудь библиотеку, написанную командой разработчиков на каком-нибудь другом языке программирования. Предположим, что существует такой язык, как Visual CatPro, в котором слова `this`, `else`, `foreach` не являются зарезервированными. И автор библиотеки, не подозревая о языке C#, спокойно использовал эти слова в качестве названий классов, функций и т. п. Если вы используете эту библиотеку, то вам придется проставить символ `@` перед ключевым словом, как в лис-

тинге 1.4 он стоит перед `this`, чтобы иметь возможность работать с библиотекой без проблем.

**Листинг 1.4. Использование символа @ в зарезервированных словах**

```
using System;
using строка = System.String;

class MainClass
{
     static void Main()
    {
        строка ИмяКота = "Рыжик";
        int @this = 5;
        Console.WriteLine("Моему коту " + @this + " лет");
    }
}
```

## Как узнать, присвоено ли переменной значение

Чтобы узнать, присвоено ли переменной какое-нибудь значение, можно воспользоваться новым типом `Nullable`, который появился в .NET Framework 2.0. В листинге 1.5 приведен пример объявления переменной как `Nullable` и проверки ее значения.

**Листинг 1.5. Проверка на присвоение значения переменной**

```
Nullable<bool> bFlag = null;

// Снимите комментарий со следующей строки,
// чтобы присвоить переменной значение
//bFlag = true;

if (bFlag.HasValue)
    this.Text = "Переменная bFlag имеет значение: " + bFlag.Value;
else
    MessageBox.Show("Переменной bFlag не присвоено никакого значения");
```

Существует и короткая формы записи для `Nullable`:

```
bool? bFlag = null;  // альтернативный вариант
```

Объявление переменной как `Nullable` делает доступными такие свойства, как `HasValue` и `Value`, которые помогают узнать, было ли присвоено переменной значение. Для использования `Nullable` подходят все стандартные типы переменных, см. листинг 1.6.

**Листинг 1.6. Проверка на присвоение значения переменным разных типов**

```
int? i = 10;                // integer
double? dl = 3.14;          // double
bool? flag = null;          // bool
char? letter = 'a';         // символ
int?[] arr = new int?[10];  // массив
```

# Как это назвать?

Правильный выбор наименований для различных объектов — залог хорошей читаемости вашего кода. Не думайте, что читать его придется другим людям, и что нет смысла облегчать им жизнь. Если через годик после написания кода вы заглянете в него, то обнаружите, что вы уже и сами чувствуете себя, будто код написал кто-то другой. Поэтому придерживайтесь определенных правил, хотя бы для собственного удобства.

## Какая разница между *string* и *System.String*?

Некоторые начинающие программисты смущаются, когда видят в коде либо `string`, либо `String`, памятуя, что язык C# различает регистр. Но на самом деле разницы между этими словами никакой. Это дело привычки. Одни программисты (например, Ч. Петцольд) привыкли использовать подход C# и применяют слово `string`, а другие (например, Д. Рихтер) призывают придерживаться синтаксиса .NET Framework. Вот небольшая таблица соответствий:

| | |
|---|---|
| `string` | `System.String` |
| `short` | `System.Int16` |
| `ushorh` | `System.Uint16` |

Только ни в коем случае не используйте сочетание `System.string`, потому что класса `string` в пространстве имен `System` не существует.

## Выберите свои правила наименования

Microsoft призывает программистов использовать определенные правила наименования при создании проектов. В документации MSDN можно найти рекомендации по этому вопросу. Также внутри компании, в которой вы работаете, могут придерживаться собственных правил. Но в любом случае не воспринимайте рекомендации как догму. Старайтесь выработать свою схему именования в проектах, с которой вам будет удобно работать. Например, у меня осталась привычка с Visual Basic 6.0 давать префиксы в именах элементов управления. Например, кнопки начинаются с префиксов `but`, а текстовые поля с префикса `txt`. Когда я набираю в редакторе кода префикс и затем нажимаю комбинацию клавиш <Ctrl>+<Пробел>, то IntelliSence выводит список всех имеющихся кнопок или текстовых полей в проекте. Но желательно придерживаться и общепринятых правил.

## Правила для названий классов и методов

Обычно классы представляют объекты, а методы — действия. Поэтому при наименовании классов или методов придерживайтесь следующего правила. Для наименования классов используйте имена существительные — Cat, Machine, Girl. При создании методов используйте глаголы в следующей нотации — MoveLeft, ShowMessage и т. д. В этом случае вам и другим программистам будет проще ориентироваться в коде.

# Советы по созданию эффективных и масштабируемых приложений

В одном из номеров журнала MSDN Magazine автор Кен Спенсер в ответ на вопрос от одного читателя приводит несколько полезных советов и рекомендаций по созданию эффективного и масштабируемого приложения. Я приведу несколько таких полезных советов, которые мне показались очень интересными, в моей интерпретации.

## Сопряжение

Сопряжение определяет, насколько тесна связь между двумя кусками кода в вашей программе? Как показывает практика, чем слабее такая связь, тем

лучше. Например, если на форме расположен элемент управления `txtInfo`, то в блоке кода, который отвечает за базовую функциональность, не должно быть прямой связи с данным текстовым полем. Вместо этого код должен получать значение или ссылку на текстовое поле.

Каждый блок кода должен быть узкоспециализированным. Не стоит писать универсальную функцию-комбайн `UpdateInfoSendMailGetSMS`, которая работает по принципу все-в-одном. Лучше написать три отдельные функции `UpdateInfo`, `SendMail` и `GetSMS`. Принцип "разделяй и властвуй" как нельзя лучше подходит в программировании.

## Наследование

Для поддержки расширяемости используйте механизм наследования. Платформа .NET Framework легко позволяет изменять поведение элемента управления, расширить его функциональность, создав новый элемент наследованием от существующего элемента. Не забывайте о наследовании форм. Однажды создав базовую форму со стандартными элементами (логотипами, общими кнопками, меню и т. п.), вы можете создавать новые формы простым наследованием от базовой. Тем самым вы упрощаете разработку и сопровождение приложения, уменьшаете объем кода, который приходится писать.

## Минимизация кода

Старайтесь, чтобы в пользовательском интерфейсе было минимум кода. Например, обработчик события `Click` для кнопки должен содержать лишь несколько строк кода. Если код становится слишком сложным или приобретает универсальный характер, то вынесите его в виде функции в отдельный модуль. В этом случае вы получите логически выстроенную архитектуру, которая обеспечит вам возможность повторного использования кода.

## Экономия ресурсов

Продумайте заранее, какие ресурсы понадобятся форме. Например, вместо формы с сотнями элементов управления используйте формы с меньшей функциональностью, или разбейте ее на несколько форм. Кроме того, можно добавлять или удалять элементы управления динамически, тем самым сокращая число статических элементов.

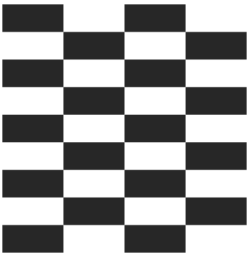
## Создание автоматически обновляемых приложений

Чтобы облегчить развертывание вашего приложения внутри компании, используйте возможности автоматического обновления программы. Windows Forms позволяет легко написать приложение, которое в автоматическом режиме проверяет наличие обновлений и само скачивает их с сервера. При этом подобные приложения следует разделять на небольшие части-компоненты, что облегчит процесс обновления.

# Заключение

Итак, мы для разминки рассмотрели самые простые примеры. Надеюсь, вы настроились на нужный лад и готовы к знакомству с более сложными примерами.

## Глава 2



# Строки, даты, числа

В этой главе речь пойдет о базовых объектах, работать с которыми приходится практически в любом приложении.

## Строки

Строки встречаются практически во всех приложениях, поэтому хорошее знание всех строковых методов и свойств необходимо любому программисту. Рассмотрим несколько советов. Необходимо помнить, что строки представлены классом `System.String` (или `string` в нотации C#) и любые операции, связанные со строками, на самом деле не изменяют саму строку, а создают новую строку. Поэтому во многих случаях лучше использовать класс `StringBuilder`, который в большей степени оптимизирован для манипуляции со строками.

### Простейшие операции со строками

Для вставки одной строки в другую используется метод `Insert`, проиллюстрированный в листинге 2.1.

**Листинг 2.1. Вставка одной строки в другую строку**

```
string partBookTitle = "C#.советы";
string insertText = "Народные ";
string bookTitle = partBookTitle.Insert(3, insertText);
MessageBox.Show(bookTitle);
```

Для удаления подстроки из заданной строки используется метод `Remove`. Если вы хотите удалить подстроку с указанной позиции до конца строки, то достаточно указать индекс нужного символа, как это сделано в листинге 2.2.

**Листинг 2.2. Удаление подстроки из заданной строки**

```
string bookTitle = "C#.Народные советы";
// Удаляем подстроку с третьей позиции
bookTitle = bookTitle.Remove(2);
MessageBox.Show(bookTitle);
```

Для извлечения части строки из заданной строки используется метод `Substring`, применение которого показано в листинге 2.3.

**Листинг 2.3. Извлечение подстроки из заданной строки**

```
string bookTitle = "C#.Народные советы";
// Извлекаем подстроку с шестой позиции с размером в три символа
bookTitle = bookTitle.Substring(5, 3);
MessageBox.Show(bookTitle);
```

> ***Примечание***
>
> Visual Basic.NET имеет в своем арсенале также такие методы и свойства, как `Left`, `Right`, `Len` и другие, унаследованные от Visual Basic 6.0. Вполне возможен вариант, что вам может попасться код, который использует эти устаревшие конструкции. Необходимо избавляться от подобных строчек кода, которым есть достойная альтернатива среди классов .NET Framework.

## Входит ли строка в другую строку?

Метод `IndexOf`, проиллюстрированный в листинге 2.4, позволяет определить место вхождения подстроки в заданную строку.

**Листинг 2.4. Определение вхождения подстроки в заданной строке**

```
string str1 = "око";
string str2 = "Царь-колокол";
int i = str2.IndexOf(str1);
// Проверяем, входит ли строка око в слово Царь-колокол
if(i >= 0) MessageBox.Show(str1 + " входит в строку " + str2);
```

Также можно воспользоваться регулярным выражением, подключив пространство имен `System.Text.RegularExpressions`. В этом случае используется метод `Regex.IsMatch`. А если вы хотите написать собственный метод для нахождения строки в другой строке (для общего развития), то обратите внимание на статью "Find a string inside another string" на сайте **http://dirtydogstink.com/blog/2007/02/16/how2FindAStringInsideAnotherString.aspx**, чтобы не изобретать заново велосипед.

## Преобразование строки в число

Не менее распространенная задача — преобразовать строку в число. В этом случае, как показано в листинге 2.5, можно использовать метод `Parse`.

**Листинг 2.5. Преобразование строки в число**

```
string tankman = "4";
string dog = "1";
textBox1.Text = tankman + dog; // получим 41
int all = int.Parse(tankman) + int.Parse(dog);
textBox1.Text += Environment.NewLine + all.ToString();  // получим 5
```

Как видите, сначала мы пытаемся сложить две строки, чтобы узнать численность экипажа танка из фильма "Четыре танкиста и собака". Результат, равный 41, нас явно не устроит. Поэтому сначала преобразуем строки в числа, а затем снова попытаемся сложить две переменные.

Существует еще один способ преобразования строки в число — методы класса `Convert`.

## Вставка специального символа

Несмотря на то, что на клавиатуре имеется более 100 клавиш, некоторые символы на клавиатуре отсутствуют. Например, к разряду таких символов можно отнести знаки копирайта, знак зарегистрированной торговой марки, символы некоторых валют и т. д. В стандарте Unicode знак копирайта имеет десятичный код 169. В листинге 2.6 показано, как, используя класс `Convert`, можно преобразовать код нужного символа и получить нужный символ как часть строки, который затем можно вывести в текстовом поле (рис. 2.1).

**Листинг 2.6. Вставка специального символа**

```
int charCode = 169;
char specialChar = Convert.ToChar(charCode);
textBox1.Text = specialChar.ToString();
```

Можно использовать и шестнадцатеричное значение кода Unicode. Например, для вывода знака зарегистрированной торговой марки используем пример, приведенный в листинге 2.7.

**Листинг 2.7. Вставка символа торговой марки**

```
// Unicode-код для торговой марки
int charCode = 0x00AE;
char specialChar = Convert.ToChar(charCode);
textBox1.Text += specialChar.ToString();
```

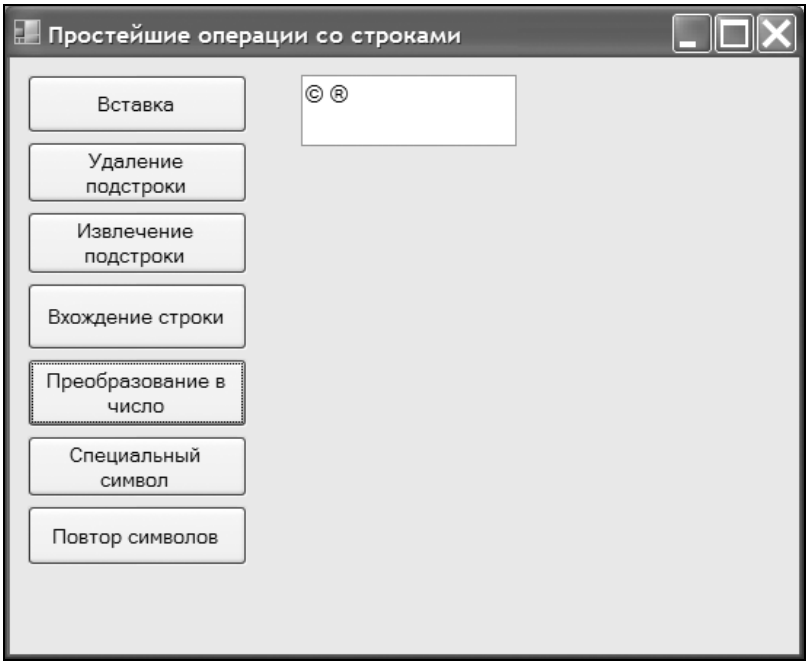



**Рис. 2.1.** Вывод специальных символов в текстовом поле

## Создание строки из повторяющихся символов

Для создания строки, состоящей из одинаковых символов, достаточно (листинг 2.8) использовать простую конструкцию.

**Листинг 2.8. Создание строки из повторяющихся символов**

```
// Создаем строку из 5 звездочек
System.String FiveStars = new System.String('*', 5);
textBox1.Text = FiveStars;
```

Все остальные методы для работы со строками вы можете изучить самостоятельно. Комбинируя строковые методы, можно добиться некоторых интересных эффектов.

## Метод *String.Format*

Не забывайте о возможности использовать метод `String.Format` при выводе строк для представления текста в нужном виде. Например, чтобы вывести строчку "Стоимость BMW равна 12 000 р.", можно использовать код, показанный в листинге 2.9.

**Листинг 2.9. Использование метода `String.Format`**

```
private void butFormatStr_Click(object sender, EventArgs e)
{
    string AutoName;
    AutoName = "BMW";
    textBox1.Text =
        String.Format("Стоимость {0} равна {1:0.0;c}", AutoName, 12000);
}
```

## Преобразование строки в объект *Color*

Если у вас возникнет необходимость преобразовать строку с названием цвета в сам объект `Color`, то воспользуйтесь классами `TypeDescriptor` и `TypeConverter`, который предоставляет унифицированный способ конвертирования типов значений в другие типы. Например, мы хотим преобразовать цвет `Color.Blue` в строку (или наоборот, преобразовать строку `"Green"` в объект `Color`). Делается это, как показано в листинге 2.10.

**Листинг 2.10. Преобразование строки в объект `Color` и обратно**

```
private void butColorName_Click(object sender, EventArgs e)
{
    // Задаем цвет
    Color clr = Color.Blue;
    // Получим имя выбранного цвета
    textBox1.Text =
         (TypeDescriptor.GetConverter(clr).ConvertToString(clr));

    // Обратная задача. Конвертируем название цвета Green в объект Color
    clr = (Color)TypeDescriptor.GetConverter(
                              typeof(Color)).ConvertFromString("Green");
    // Закрашиваем форму в выбранный цвет
    this.BackColor = clr;

}
```

## Проверка строки на пустоту

Существует несколько способов проверить строку на пустоту:

- `if (myString == "") // пустая строка;`
- `if (myString == String.Empty) // пустая строка;`
- `if (myString.Length == 0) // строка с нулевой длиной;`
- `if (String.Equals(myString, String.Empty)).`

Ян Нельсон (Ian Nelson) в своем блоге **http://ianfnelson.com/blog/archive/2004/07/30/171.aspx** не поленился и подсчитал, сколько времени занимает каждый из этих вариантов при обработке 50 миллионов итераций. Результаты вы можете посмотреть на сайте, но самым быстрым вариантом стал третий способ. Но данный способ не учитывает, что строка может принимать значение `null`. Эксперименты проводились в 2004 году. С появлением .NET Framework 2.0 класс `System.String` обзавелся новым статическим методом `IsNullOrEmpty` (листинг 2.11), который позволяет быстро и просто проверить строку — пустая она или имеет значение `null`.

**Листинг 2.11. Проверка строки на пустоту**

```
public void SayHello(string name)
{
```

```
    if ( string.IsNullOrEmpty(name) )
         throw new ArgumentNullException("name");
    MessageBox.Show( string.Concat("Hello, ", name) );
}
```

## Переворачиваем строку

Иногда встречается задача перевернуть строку наоборот. В Visual Basic 6.0 появилась удобная строковая функция `StrReverse`, которая позволяла быстро и удобно перевернуть строку. Разработчики Visual Studio оставили эту функцию для программистов Visual Basic .NET. Таким образом, вам нужно установить ссылку на пространство имен `Microsoft.VisualBasic` и воспользоваться функцией по своему назначению (листинг 2.12).

**Листинг 2.12. Переворачиваем строку при помощи функции Visual Basic**

```
using Microsoft.VisualBasic;

private void butReverseVB_Click(object sender, EventArgs e)
{
    // Взять текст из текстового поля
    // Например, А роза упала на лапу Азора
    string myString   = textBox1.Text;
    // Используем встроенную функцию Visual Basic
    textBox1.Text = Strings.StrReverse(myString);
}
```

Если вы по каким-то причинам не хотите пользоваться библиотекой Visual Basic, можно написать собственную процедуру переворачивания строки, приведенную в листинге 2.13.

**Листинг 2.13. Переворачиваем строку при помощи C#**

```
public static string ReverseString(string str)
{
    // Проверка на непустоту строки.
    if(string.IsNullOrEmpty(str))
    {
        return str;
    }
```

```
    // Создадим объект StringBuilder с нужной длиной.
    StringBuilder revStr = new StringBuilder(str.Length);
    // Перебираем в цикле все символы
    // и присоединяем каждый символ к StringBuilder
    for (int count = str.Length - 1; count > -1; count--)
    {
        revStr.Append(str[count]);
    }
    // Возвращаем перевернутую строку
    return revStr.ToString();
}

private void butReverseCS_Click(object sender, EventArgs e)
{
    textBox1.Text = ReverseString(textBox1.Text);
}
```

***Примечание***

Примеры с простейшими операциями со строками вы можете найти в папке SimpleStrings на прилагаемом диске.

## Сжатие длинных имен файлов

Если вам часто приходится работать со строками, которые представляют собой длинные полные пути к файлам, то возможно вам пригодится следующий пример. С помощью функции Windows API `PathCompactPathEx` можно заменить часть длинного пути к файлу многоточием, например, как это сделано в листинге 2.14.

**Листинг 2.14. Замена длинного пути многоточием**

```
using System.Runtime.InteropServices;

[DllImport("shlwapi.dll", CharSet = CharSet.Auto, SetLastError = true)]
private static extern bool PathCompactPathEx(
    System.Text.StringBuilder pszOut,
    string pszSrc,
    Int32 cchMax,
    Int32 dwFlags);
```

```
private void button1_Click(object sender, EventArgs e)
{
    // длинный путь к файлу
    string strPathFile =
              "c:/program files/My SuperProgram/skins/sample.txt";
    StringBuilder sb = new StringBuilder(260);

    // оставляем 20 символов, остальное заменяем многоточием
    bool b = PathCompactPathEx(sb, strPathFile, 20+1, 0);

    // Выводим результат в текстовое поле
    textBox1.Text = sb.ToString();
}
```

***ПРИМЕЧАНИЕ***

Пример со сжатием строки находится в папке CompactPathDemo на прилагаемом диске.

## Печатающийся текст

Чтобы создать видимость того, что текст печатается на печатной машинке, нужно использовать таймер, с помощью которого через определенный промежуток времени мы считываем все символы текста с первой буквы до текущей и выводим их на экран. Счетчик текущего количества букв постоянно обновляется (листинг 2.15), что позволяет повторять эффект печатающегося текста бесконечно, пока пользователь не остановит этот процесс.

**Листинг 2.15. Эффект печатающегося текста**

```
private void button1_Click(object sender, EventArgs e)
{
    if (button1.Text == "Старт")
    {
        timer1.Enabled = true;
        button1.Text = "Стоп";
    }
    else
    {
        timer1.Enabled = false;
```

```
        button1.Text = "Старт";
    }
}

public static int counter = 0;

private void timer1_Tick(object sender, EventArgs e)
{
    string typingText = "C#.Народные советы";

    this.Text = typingText.Substring(0,counter);
    counter++;

    if (counter > typingText.Length)
        counter = 0;
}
```

## Бегущая строка

Второй распространенный эффект со строками — создание бегущей строки. Алгоритм создания эффекта бегущей строки заключается в следующем — из исходной строки удаляем один символ слева и добавляем его с правой стороны текста.

**Листинг 2.16. Создание бегущей строки**

```
private void butScroll_Click(object sender, EventArgs e)
{
    timer2.Enabled = true;
}

// Исходная строка. Для большего удобства добавлено несколько пробелов
// в конец строки
private string scrollText = "C#.Народные советы    ";

private void timer2_Tick(object sender, EventArgs e)
{
    // Удаляем один символ слева и прибавляем его с правой стороны
    scrollText = scrollText.Substring(1,
```

```
            (scrollText.Length - 1)) + scrollText.Substring(0, 1);
    this.Text = scrollText;
}
```

***Примечание***

Примеры находятся в проекте StringsFX на прилагаемом диске.

## Как соединять строки

Во многих книжках пишут, что для сцепления строк нужно использовать класс `StringBuilder`, как самый эффективный способ.

**Листинг 2.17. Соединение строк при помощи `StringBuilder`**

```
StringBuilder sb = new StringBuilder();
sb.Append("строка 1");
sb.Append("строка 2");
this.Text = sb.ToString();
```

Однако, на самом деле, это не всегда самый лучший выбор. В одном из выпусков журнала "MSDN Magazine" была статья, в которой говорилось, что существуют случаи, когда использование класса `StringBuilder` для каждой операции сцепления является не самым быстрым решением. Использование класса `StringBuilder` наиболее эффективно, если сцепляется много строк, если число строк неизвестно или если строки длинные. Однако если в вашей программе сцепляются лишь несколько строк, или если они относительно коротки, то лучшим выбором часто является использование стандартной операции сцепления. В этих случаях можно использовать либо стандартную конструкцию сцепления используемого языка (для языка C# это символ +), либо метод `String.Concat`. Такие способы обеспечивают сравнимую производительность. Дело в том, что при использовании класса `StringBuilder` происходят дополнительные затраты ресурсов, которые не возникают при использовании стандартных строк. К тому же, класс `StringBuilder`, в конечном счете, необходимо все равно преобразовывать в строку, что опять приводит к дополнительным расходам. Следовательно, если вы занимаетесь оптимизацией вашего приложения, то вам необходимо убедиться в том, что используемые методы сцепления строк себя оправдывают. В указанной статье рассматривается пример сцепления переменного количества строк длиной 25 символов каждая и показываются графики зависимости времени, не-

обходимого для сцепления нескольких двадцатипятисимвольных строк с объектом класса `StringBuilder` и со строкой, от их количества. Результаты показывают, что если число сцеплений больше семи, лучше использовать класс `StringBuilder`, а если сцепляемых строк меньше, обычные способы соединения строк дают лучшую производительность.

> ***Примечание***
>
> Более подробную информацию вы можете прочитать в статье "CLR Inside Out" в январском номере 2006 года журнала "MSDN Magazine". Существует также русскоязычная онлайн-версия статьи "CLR вдоль и поперек" на сайте **http://msdn.microsoft.com:80/msdnmag/issues/06/01/CLRInsideOut/default.aspx?loc=ru**.

## Что лучше: *Parse* или *TryParse?*

В статье из предыдущего примера также сравниваются возможности методов `Parse` и `TryParse`. В .NET Framework 2.0 появился новый метод для базовых типов `TryParse`, который похож на метод `Parse`. Разница в их поведении заключается в том, что метод `Parse` при неудачной проверке данных вызывает исключение, а метод `TryParse` просто переходит к инструкции `else`. Нас интересует, что лучше использовать в программах для повышения производительности. Оказывается, что при успешном анализе значения строки `someString` производительность обоих методов в точности одинакова. Таким образом, использование метода `Parse` не приводит к дополнительным затратам, если точно известно, что имеющаяся строка соответствует тому типу, в который она преобразуется. Однако производительность падает, если вам попадется хотя бы одна строка, которая не может быть успешно преобразована. В этом случае при использовании метода `Parse` управление будет передано в блок обработки исключения. А исключения практически всегда замедляют работу. Если данная ситуация возникает не слишком часто, то это не очень страшно. В противном случае приходится платить значительным ухудшением производительности. В противоположность методу `Parse` метод `TryParse` при такой ситуации использует блок `if-else`. При переходе к блоку `else` производительность не страдает. Метод `TryParse` всегда работает с той же или большей производительностью, что и метод `Parse`, но разница между ними иногда может быть значительной. В статье приводятся такие цифры: в самом крайнем случае, когда ни одну строку не удается преобразовать в базовый тип, использование метода

`Parse` может занять в 150 раз больше времени, чем использование метода `TryParse`. С другой стороны, если анализируется 100 строк, и только одна из них не может быть успешно преобразована, метод `TryParse` работает всего лишь на 30 процентов быстрее. Таким образом, подтверждается общее правило, что не следует использовать исключения в качестве механизма управления логической последовательностью операций. Хотя использование метода `TryParse` более предпочтительно, если необходимо определить, соответствует ли строка другому типу данных, все же в некоторых ситуациях вместо него следует использовать метод `Parse`. Одной из них является ситуация, при которой ожидается, что анализ будет успешным, и, следовательно, неудачный анализ действительно является исключительным условием, а снижение производительности проблемы не составляет. Другой важный случай — когда необходимо определить конкретные причины неудачного преобразования, например, различить исключительные ситуации, связанные с переполнением и с неверным форматом. Для обоих примеров весьма подходящим является метод `Parse`. Исключение `FormatException`, возникающее в результате неудачи метода `Parse`, означает, что неудача была неожиданной, и случай действительно является исключительным по определению.

Еще раз советую перечитать статью, ссылка на которую дана в предыдущем разделе.

## Сравнение и сортировка строк

И, наконец, в указанной выше статье сказано несколько слов о сравнении строк. Строки в среде .NET являются довольно мощным инструментом в том смысле, что их можно сравнивать с учетом и без учета особенностей культурной среды. По умолчанию большинство строковых операций подобные особенности учитывают. Если нет необходимости сравнивать строки с учетом особенностей культурной среды, то для сравнения строк всегда следует использовать сравнение по порядковым номерам. Это же касается и сортировки строк. Экономия при использовании этого способа сортировки или сравнения получается значительной. Для класса `System.String` существуют специальные строковые методы, например, `String.CompareOrdinal`, подчеркивающие ценность такого подхода. Если вам не нужно учитывать регистр строк при сравнении, но при этом важно сохранить производительность, следует воспользоваться методом `StringComparer.OrdinalIgnoreCase`. Он работает намного быстрее, чем метод, учитывающий особенности культурной

среды, и при этом предоставляет возможность не учитывать регистр, отсутствующую во многих методах сортировки и сравнения.

# Даты

Использование дат в приложении является необходимостью для многих разработчиков программ. Вот несколько полезных советов для работы с датами.

## Как получить текущую дату

Для получения текущей даты можно использовать свойство `DateTime.Now`, что иллюстрируется листингом 2.18.

**Листинг 2.18. Получение текущей даты**

```
DateTime today = DateTime.Now;
MessageBox.Show(today.Date.ToLongDateString());
```

Обратите внимание, что мы использовали метод `ToLongDateString` вместо обычного метода `ToString`, чтобы вывести дату в длинном формате (рис. 2.2).

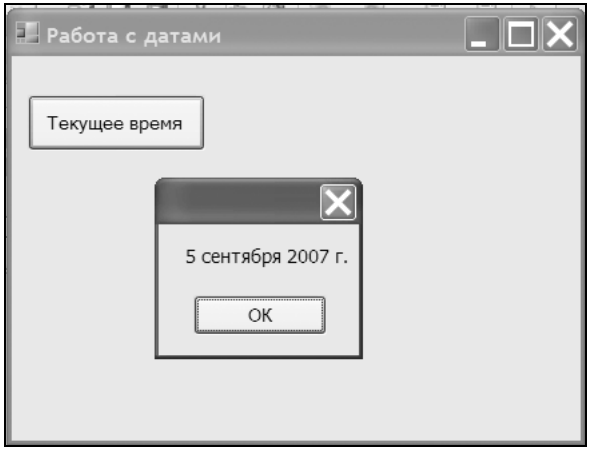



**Рис. 2.2.** Вывод даты в длинном формате

## Дата и время в разных форматах

Существует множество форматов для вывода на экран даты и времени. В листинге 2.19 я постарался собрать все знакомые мне способы форматирования, используемые в методе `ToString`.

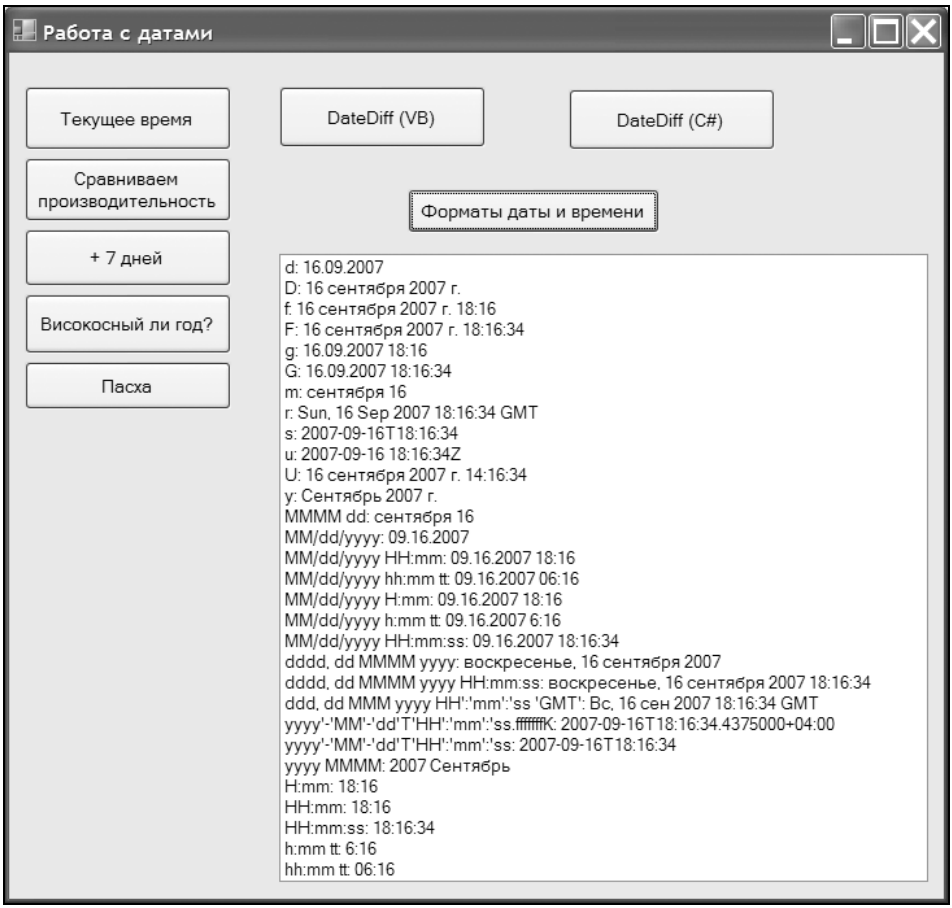



**Рис. 2.3.** Вывод даты и времени в разных форматах

**Листинг 2.19. Вывод даты и времени в разных форматах**

```
private void butDateTime_Click(object sender, EventArgs e)
{
    DateTime dt = DateTime.Now;
```

```
listBox1.Items.Clear();
listBox1.Items.Add("d: " + dt.ToString("d"));
listBox1.Items.Add("D: " + dt.ToString("D"));
listBox1.Items.Add("f: " + dt.ToString("f"));
listBox1.Items.Add("F: " + dt.ToString("F"));
listBox1.Items.Add("g: " + dt.ToString("g"));
listBox1.Items.Add("G: " + dt.ToString("G"));
listBox1.Items.Add("m: " + dt.ToString("m"));
listBox1.Items.Add("r: " + dt.ToString("r"));
listBox1.Items.Add("s: " + dt.ToString("s"));
listBox1.Items.Add("u: " + dt.ToString("u"));
listBox1.Items.Add("U: " + dt.ToString("U"));
listBox1.Items.Add("y: " + dt.ToString("y"));

listBox1.Items.Add("MMMM dd: " + dt.ToString("MMMM dd"));
listBox1.Items.Add("MM/dd/yyyy: " + dt.ToString("MM/dd/yyyy"));
listBox1.Items.Add("MM/dd/yyyy HH:mm: " +
            dt.ToString("MM/dd/yyyy HH:mm"));
listBox1.Items.Add("MM/dd/yyyy hh:mm tt: " +
            dt.ToString("MM/dd/yyyy hh:mm tt"));
listBox1.Items.Add("MM/dd/yyyy H:mm: " +
            dt.ToString("MM/dd/yyyy H:mm"));
listBox1.Items.Add("MM/dd/yyyy h:mm tt: " +
            dt.ToString("MM/dd/yyyy h:mm tt"));
listBox1.Items.Add("MM/dd/yyyy HH:mm:ss: " +
            dt.ToString("MM/dd/yyyy HH:mm:ss"));

listBox1.Items.Add("dddd, dd MMMM yyyy: " +
            dt.ToString("dddd, dd MMMM yyyy"));
listBox1.Items.Add("dddd, dd MMMM yyyy HH:mm:ss: " +
            dt.ToString("dddd, dd MMMM yyyy HH:mm:ss"));
listBox1.Items.Add("ddd, dd MMM yyyy HH':'mm':'ss 'GMT': " +
            dt.ToString("ddd, dd MMM yyyy HH':'mm':'ss 'GMT'"));

listBox1.Items.Add("yyyy'-'MM'-'dd'T'HH':'mm':'ss.fffffffK: " +
            dt.ToString("yyyy'-'MM'-'dd'T'HH':'mm':'ss.fffffffK"));
listBox1.Items.Add("yyyy'-'MM'-'dd'T'HH':'mm':'ss: " +
            dt.ToString("yyyy'-'MM'-'dd'T'HH':'mm':'ss"));
listBox1.Items.Add("yyyy MMMM: " + dt.ToString("yyyy MMMM"));
```

```
    listBox1.Items.Add("H:mm: " + dt.ToString("H:mm"));
    listBox1.Items.Add("HH:mm: " + dt.ToString("HH:mm"));
    listBox1.Items.Add("HH:mm:ss: " + dt.ToString("HH:mm:ss"));
    listBox1.Items.Add("h:mm tt: " + dt.ToString("h:mm tt"));
    listBox1.Items.Add("hh:mm tt: " + dt.ToString("hh:mm tt"));
}
```

Запустив этот пример, вы увидите, как по-разному можно представить одну и ту же дату, используя различные комбинации (рис. 2.3).

## Как использовать дату и время в приложении

Дату и время можно использовать как местные, так и в формате единого времени (UTC). Оказывается, свойство `UtcNow`, обозначающее всеобщее скоординированное время, работает намного лучше свойства `Now`. Безусловно, при выводе даты и времени на экран пользователя нет смысла применять свойство `UtcNow`, но при сравнении двух значений времени и даты с целью узнать, какое событие произошло раньше, или для выполнения других вычислений, когда не нужно учитывать особенности культурной среды, выбор свойства `UtcNow` является предпочтительным. Таким образом, дату и время лучше хранить как значения формата UTC, пока не понадобится извлечь их в каком-то локализованном формате. Даже не думая о производительности, удобно хранить значения даты и времени в формате единого времени UTC, поскольку это позволяет точно сравнивать данные нескольких часовых поясов без влияния перехода на летнее и зимнее время. А в следующем примере (листинг 2.20) мы увидим и разницу в производительности.

**Листинг 2.20. Измерение производительности двух свойств**

```
// Измеряем производительность для UtsNow
DateTime dt = DateTime.UtcNow;
Stopwatch sw = new Stopwatch();
for (int i = 0; i < 10; i++) {
    sw.Start();
    for (int j = 0; j < 100000; j++) {
        if (DateTime.UtcNow == dt)
        {
            /* do action */
```

```
        }
    }
    MessageBox.Show(sw.ElapsedTicks.ToString());
}

// Аналогичная проверка производительности для Now
DateTime dt = DateTime.Now;
Stopwatch sw = new Stopwatch();
for (int i = 0; i < 10; i++) {
    sw.Start();
    for (int j = 0; j < 100000; j++) {
        if (DateTime.Now == dt)
        {
            /* do action */
        }
    }
    MessageBox.Show(sw.ElapsedTicks.ToString());
}
```

Запустите пример, чтобы проверить разницу в производительности. Нужно просто заменить вызов `UtcNow` на вызовы `Now`, чтобы получить результат для местного времени. Вы увидите, что, действительно, метод `UtcNow` работает быстрее, чем метод `Now`.

## Сложить и вычесть временной интервал из дат

Для сложения или вычитания какого-нибудь временного интервала можно использовать многочисленные методы с приставкой `Add`. Например, с помощью метода `AddDays` можно добавить или убавить заданное число дней из выбранной даты (листинг 2.21).

**Листинг 2.21. Добавление 7 дней к текущей дате**

```
DateTime curdate = DateTime.Now;
// Прибавляем семь дней к текущей дате
DateTime mydate = curdate.AddDays(7);
MessageBox.Show(mydate.ToShortDateString());
```

# Вычисление разницы между датами

Достаточно распространенная задача в программировании — вычислить число дней между датами. Рассмотрим несколько способов для решения этой проблемы.

## Метод *DateDiff*

Программисты на Visual Basic имеют в своем арсенале функцию `DateDiff`, которая была еще в Visual Basic 6.0. Никто нам не мешает создать ссылку на сборку Microsoft.VisualBasic.dll и воспользоваться этой функцией в своих целях. В меню **Project** выбираем пункт **Add Reference** и в диалоговом окне ищем элемент **Microsoft.VisualBasic.dll**. Далее в редакторе кода импортируем пространство имен Microsoft.VisualBasic и пишем следующий код (листинг 2.22).

**Листинг 2.22. Вычисляем количество дней между датами с помощью функции Visual Basic `DateDiff`**

```
using Microsoft.VisualBasic;

private void butDateDiffVB_Click(object sender, EventArgs e)
{
    DateTime date1 = new DateTime(1945, 5, 9);  // Первая дата
    DateTime date2 = new DateTime(2007, 8, 21); // Вторая дата

    // Вычисляем разницу в днях при помощи перечисления DateInterval.Day
    long diff = DateAndTime.DateDiff(DateInterval.Day, date1, date2,
        FirstDayOfWeek.Monday,FirstWeekOfYear.System);

    // Выводим результат в заголовке формы
    this.Text = diff.ToString();
}
```

## Второй способ — собственный метод *DateDiff*

Среди программистов C# использовать пространство имен `VisualBasic` не принято. Чувство собственного достоинства не позволяет им опускаться до языка для начинающих ☺. Ладно, напишем свой собственный метод `DaysDiff`, который вычислит разницу между датами при помощи метода `Subtract` (листинг 2.23).

**Листинг 2.23. Собственный метод `DateDiff`**

```
// Наш собственный метод вычисления разницы в датах
public static int DaysDiff(DateTime date1, DateTime date2)
{
    return date1.Subtract(date2.Date).Days;
}

private void butDateDiffCS_Click(object sender, EventArgs e)
{
    string howdays = DaysDiff(DateTime.Today,
                        new DateTime(1966, 07, 27)).ToString();
    MessageBox.Show("Со дня моего дня рождения прошло: " + howdays);
}
```

Я решил проверить, сколько дней прошло со дня моего рождения до сегодняшнего дня, и получил красивое число. Оказывается, на момент написания этих строк я прожил ровно 15 000 дней (рис. 2.4).

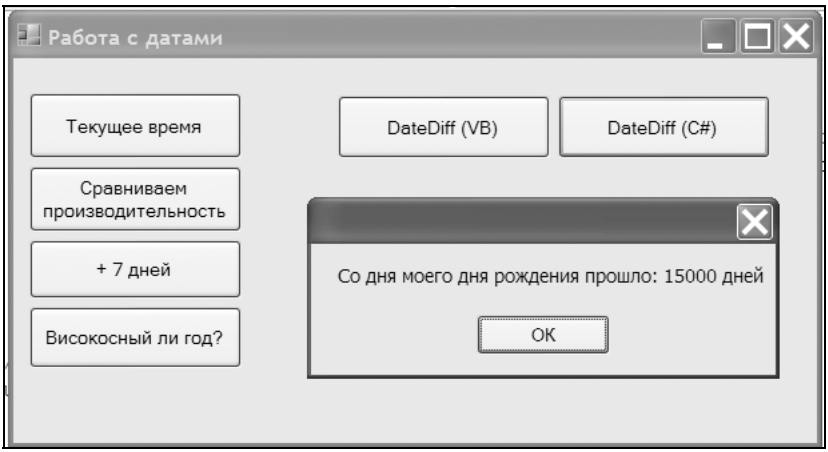



**Рис. 2.4.** Сколько дней прошло со дня моего рождения

## Как определить, является ли год високосным?

Для определения, является ли указанный год високосным годом, используйте функцию `IsLeapYear`, которая возвращает значение `true`, если год является

високосным, и значение `false`, если год таким не является. Проверим в следующем примере, приведенном в листинге 2.24, является ли текущий год високосным.

**Листинг 2.24. Проверка на високосный год**

```
bool leapYear = DateTime.IsLeapYear(DateTime.Now.Year);
MessageBox.Show(
    String.Format("{0} является високосным годом: {1}",
    DateTime.Now.Year, leapYear));
```

Также поищите в документации материал о функциях `IsLeapDay` и `IsLeapMonth`, которые помогут определить, являются ли указанные день или месяц високосными.

## Вычисление даты католической Пасхи

Как вы знаете, праздник Пасхи каждый год приходится на разные числа. Существуют специальные правила для вычисления иудейской, католической и православной Пасхи. На сайте **http://snippets.dzone.com/posts/show/765** я нашел алгоритм, вычисляющий дату католической Пасхи, реализованный в листинге 2.25.

**Листинг 2.25. Вычисление даты католической Пасхи**

```
/// <summary>
/// Алгоритм для вычисления католической Пасхи
/// http://snippets.dzone.com/posts/show/765
/// </summary>
/// <param name="year">Год</param>
/// <returns>Пасха</returns>
public static DateTime EasterDate(int year)
{
    int Y = year;
    int a = Y % 19;
    int b = Y / 100;
    int c = Y % 100;
    int d = b / 4;
    int e = b % 4;
    int f = (b + 8) / 25;
    int g = (b - f + 1) / 3;
```

```
    int h = (19 * a + b - d - g + 15) % 30;
    int i = c / 4;
    int k = c % 4;
    int L = (32 + 2 * e + 2 * i - h - k) % 7;
    int m = (a + 11 * h + 22 * L) / 451;
    int month = (h + L - 7 * m + 114) / 31;
    int day = ((h + L - 7 * m + 114) % 31) + 1;
    DateTime dt = new DateTime(year, month, day);
    return dt;
}

private void butEaster_Click(object sender, EventArgs e)
{
    // когда была католическая Пасха в 2006 году
    MessageBox.Show(EasterDate(2006).ToLongDateString());

}
```

Более подробно об алгоритмах вычисления дат Пасхи вы можете узнать из Википедии по адресу **http://ru.wikipedia.org/wiki/Пасха**.

> ***Примечание***
>
> Примеры, иллюстрирующие работу с датами, находятся в папке DateSamples на прилагаемом диске.

# Числа

## Преобразование числа в шестнадцатеричную систему счисления

Программистам часто приходится иметь дело с шестнадцатеричной системой счисления. Преобразовать число в шестнадцатеричную строку не составит никакого труда при помощи метода `ToString` (листинг 2.26).

**Листинг 2.26. Преобразование числа в шестнадцатеричное значение**

```
int parrots = 38;
textBox1.Text = parrots.ToString("X8");
```

В этом примере получаемая строка будет дополнена нулями слева, чтобы общее число символов было равно 8. О том, как преобразовать строку обратно в число, было описано в начале главы, когда речь шла о строках.

## Как перевести число в двоичную систему счисления?

Метод `Convert.ToString(value, 2)` позволяет преобразовывать число в двоичную строку. Например, в листинге 2.27 показано, что число 4 при преобразовании примет значение "100".

**Листинг 2.27. Перевод числа в двоичную систему счисления**

```
int myvalue = 4;
textBox1.Text = Convert.ToString(myvalue, 2);  // возвратит 100
```

## Как перевести число в восьмеричное или шестнадцатеричное представление?

На самом деле предыдущий пример является неполным. Метод `Convert.ToString` позволяет преобразовать число не только в двоичную, но и в восьмеричную или шестнадцатеричную строку (листинг 2.28).

**Листинг 2.28. Перевод числа в различные системы счисления**

```
int myValue = 365;
// Преобразуем в восьмеричное значение
MessageBox.Show(Convert.ToString(myValue, 8));

// Преобразуем в шестнадцатеричное значение
MessageBox.Show(Convert.ToString(myValue, 16));
```

## Является ли выражение числом?

В составе библиотеки run-time Visual Basic имеется функция `IsNumeric`, знакомая еще программистам Visual Basic 6.0. Эта функция позволяет узнать, можно ли считать заданное выражение числом, как показано в листинге 2.29. Подключите эту функцию к своему проекту. Для этого в меню проекта выбе-

рите **Project** | **Add Reference** | вкладка .**NET** | элемент списка **Microsoft.VisualBasic**.

**Листинг 2.29. Проверка, является ли выражение числом, при помощи Visual Basic**

```
using Microsoft.VisualBasic;

private void butIsNumericVB_Click(object sender, EventArgs e)
{
    // введите в текстовое поле любое число или слово
    string numstring = textBox1.Text;
    bool bResult1;
    bResult1 = Information.IsNumeric(numstring);
    MessageBox.Show("Является ли " + numstring + " числом: " + bResult1);
}
```

## Создание собственной функции *IsNumeric* на C#

Если мы не хотим пользоваться возможностями языка Visual Basic, значит, нам придется писать свою функцию на чистом C#.

**Листинг 2.30. Проверка на число при помощи C#**

```
// Пишем собственную функцию IsNumeric на чистом C#
static bool IsNumeric(object Expression)
{
    // Возвращаемое значение
    bool isNum;

    // Переменная, используемая в качестве параметра
    // в методе TryParse
    double retNum;

    // Метод TryParse конвертирует строку в заданный стиль
    // и локальный формат.
    isNum = Double.TryParse(Convert.ToString(Expression),
        System.Globalization.NumberStyles.Any,
        System.Globalization.NumberFormatInfo.InvariantInfo, out retNum);
```

```
    return isNum;
}

private void butIsNumericCS_Click(object sender, EventArgs e)
{
    MessageBox.Show("Является ли " + textBox1.Text +
                            " числом: " + IsNumeric(textBox1.Text));
}
```

Для проверки, является ли выражение числом, введите любой текст, состоящий только из букв, только из цифр или смешанный вариант, и нажмите на любую из двух кнопок с текстом **Это число?**. И вы получите правильный ответ.

> ***ПРИМЕЧАНИЕ***
>
> Вариант с созданием собственной функции `IsNumeric` на C# был описан в статье Базы Знаний Microsoft, которую вы можете прочитать по адресу **http://support.microsoft.com/kb/329488/**.

## Создание уникального идентификатора

Вам нужно создать новый глобальный уникальный идентификатор (Globally Unique Identifier, GUID)? GUID является целым числом, состоящим из 128 двоичных разрядов. Главная особенность данного числа — его уникальность. Можно утверждать, что любое сгенерированное число GUID больше нигде не повторится на любом компьютере. Значения GUID используются в серьезных приложениях для уникальной идентификации различных задач. Возможно, вам также придется использовать этот GUID в ваших проектах. В библиотеке .NET Framework имеется специальный метод `Guid.NewGuid`, позволяющий сгенерировать уникальное число (листинг 2.31).

**Листинг 2.31. Создание уникального идентификатора**

```
Guid newGuid = Guid.NewGuid();
// Уникальный идентификатор с дефисами
//MessageBox.Show(newGuid.ToString());

// Уникальный идентификатор без дефисов
textBox1.Text = newGuid.ToString("N");
```

Запустив проект, вы увидите на экране длинную строку в виде последовательности цифр, разделенных дефисами. Если вы хотите получить строку без дефисов, то используйте метод `ToString("N")`.

> ***Примечание***
>
> Примеры, иллюстрирующие работу с числами, вы найдете в проекте NumbersSamples на прилагаемом диске.

# Перечисления

В некоторых случаях удобно использовать перечисления, которые упрощают написание и чтение кода. Рассмотрим несколько примеров использования перечислений.

## Как получить все элементы перечисления

Предположим, у нас имеется абстрактное перечисление `Cats`, состоящее из имен котов. Обратите внимание, что первому элементу перечисления было присвоено значение 3, второму — значение 5, а остальные члены перечисления получают значения 6, 7 и 8 соответственно. Мы хотим получить значения всех элементов перечисления и их имена. Нам на помощь придут методы `Enum.GetNames` и `Enum.GetValues`, которые и позволяют узнать эти данные (листинг 2.32).

**Листинг 2.32. Получение имен и значений перечисления**

```
enum Cats { Рыжик = 3, Барсик = 5, Мурзик, Васька, Пушок };

private void button1_Click(object sender, EventArgs e)
{
    // Перечисляем все элементы перечисления
    string[] catNames = Enum.GetNames(typeof(Cats));

    foreach (string s in catNames)
    {
        listBox1.Items.Add(s);
    }
```

```
    // Перечисляем все значения перечисления
    int[] valCats = (int[])Enum.GetValues(typeof(Cats));

    foreach (int val in valCats)
    {
        listBox1.Items.Add(val.ToString());
    }

    MessageBox.Show(catNames[3].ToString());
    MessageBox.Show(valCats[3].ToString());
}
```

В библиотеке классов .NET Framework имеется огромное количество перечислений. Например, имеется перечисление `KnownColor`, которое содержит все цвета, зарегистрированные в системе. Если мы хотим получить список этих цветов, то используем такую же технику (листинг 2.33), которую мы использовали в предыдущем примере.

**Листинг 2.33. Получение имен всех цветов системы**

```
private void button2_Click(object sender, EventArgs e)
{
    // Получаем массив строк, содержащих все цвета в системе
    string[] allcolors =
                Enum.GetNames(typeof(System.Drawing.KnownColor));

    listBox1.Items.Clear();

    // Выводим все имена в список
    listBox1.Items.AddRange(allcolors);

    // Другой вариант
    //foreach (string s in allcolors)
    //{
    //    listBox1.Items.Add(s);
    //}
}
```

Можно осуществить обратный процесс. Предположим, у нас заполнен список названиями всех зарегистрированных в системе цветов. Мы хотим, чтобы цвет фона формы соответствовал выбранному тексту в списке. Для этого

нужно выбранный в списке текст сначала преобразовать в значение `KnownColor`, а затем передать полученное значение методу `Color.FromKnowColor`. Это проиллюстрировано в листинге 2.34.

**Листинг 2.34. Передача строки в имена перечисления**

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    Object ColorEnum;
    ColorEnum = System.Enum.Parse(typeof(KnownColor),listBox1.Text);
    KnownColor SelectedColor = (KnownColor)ColorEnum;
    this.BackColor = System.Drawing.Color.FromKnownColor(SelectedColor);
}
```

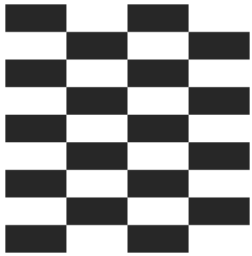
***ПРИМЕЧАНИЕ***

Пример находится в папке EnumDemo на прилагаемом компакт-диске.

# Заключение

В этой главе вы познакомились с некоторыми приемами, связанными с числами, строками, датами и перечислениями. В повседневной практике разработчику часто приходится иметь дело с этими типами данных. Поэтому, наверняка, ваша коллекция будет постоянно пополняться другими трюками.

## Глава 3



# Алгоритмы

В этой главе представлено несколько простых алгоритмов, которые могут пригодиться начинающим программистам. Примеры этой главы приводятся в качестве разминки перед знакомством с более сложными алгоритмами в последующих главах.

## Найти наименьшее и наибольшее значение из трех чисел

В составе .NET Framework имеется класс `Math`, содержащий методы `Math.Min` и `Math.Max`, которые принимают два аргумента и находят наименьшее или наибольшее значение из двух чисел. На основе этих методов можно находить наибольшее и наименьшее значения из трех чисел, как показано в листинге 3.1.

**Листинг 3.1. Нахождение наименьшего и наибольшего значения из трех чисел**

```
// Пример нахождения минимального и максимального значений из трех чисел
// при помощи методов FindMax3 и FindMin3
using System;

public class MaxMinDemo
{
    public static void Main()
    {
        int x;
        int y;
        int z;
```

```
        Console.Write("Введите первое число: ");
        x = Convert.ToInt32(Console.ReadLine());
        Console.Write("Введите второе число: ");
        y = Convert.ToInt32(Console.ReadLine());
        Console.Write("Введите третье число: ");
        z = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Наибольшее число из трех: " +
                                                FindMax3(x, y, z));
        Console.WriteLine("Наименьшее число из трех: " +
                                                FindMin3(x, y, z));
        Console.ReadLine();
    }

    // Метод для нахождения наибольшего значения из трех заданных чисел
    public static int FindMax3(int a, int b, int c)
    {
        int max;
        max = Math.Max(a, Math.Max(b, c));
        return max;
    }

    // Метод для нахождения наименьшего значения из трех заданных чисел
    public static int FindMin3(int a, int b, int c)
    {
        int min;
        min = Math.Min(a, Math.Min(b, c));
        return min;
    }
}
```

# Массив строк

Хотите объявить и инициализировать массив строк, но устали от кавычек и запятых? В самом деле, когда мы объявляем массив строк, нам приходится каждое слово окружать кавычками и отделять друг от друга запятыми. Попробуйте следующий способ (листинг 3.2), который поможет немного сэкономить ваше время.

Подключим пространство имен `System.Text.RegularExpressions` и объявим массив строк следующим образом:

**Листинг 3.2. Быстрое создание массива строк при помощи метода `Split`**

```
using System.Text.RegularExpressions;

String[] s = Regex.Split("Январь Февраль Март Апрель Май Июнь Июль Август
Сентябрь Октябрь Ноябрь Декабрь", " ");
```

Метод `Split` объекта `RegEx` возвращает массив объектов `String`, содержащий элементы из строки, переданной в первом параметре, разделенных регулярным выражением, заданным во втором параметре. Например, в нашем случае слова разделяются пробелом (можно использовать и другой символ в качестве разделителя). Таким образом, мы получили массив строк, при этом нам не пришлось ставить лишние кавычки. Полный исходный код программы, использующий этот метод, приводится в листинге 3.3.

**Листинг 3.3. Полный пример создания массива строк**

```
using System.Text.RegularExpressions;

class ScopeDemo
{
    public static void Main()
    {
        String[] s = Regex.Split("Январь Февраль Март Апрель Май Июнь
Июль Август Сентябрь Октябрь Ноябрь Декабрь", " ");
        // Выводим 12 элемент массива
        Console.WriteLine(s[11]);
    }
}
```

# Преобразование градусов в радианы и радианов в градусы

В тригонометрических функциях класса `Math` для расчетов используются радианы. Для перевода градусов в радианы нужно число градусов умножить на число π, деленное на 180.

Если вам нужно конвертировать значения в градусах в радианы, то используйте функцию, определенную в листинге 3.4.

**Листинг 3.4. Конвертируем градусы в радианы**

```
using System;

public static double ConvertDegreesToRadians(double degrees)
{
    double radians = (Math.PI / 180) * degrees;
    return radians;
}
```

Соответственно, для обратного конвертирования значений в радианах в градусы используем обратную ей функцию (листинг 3.5).

**Листинг 3.5. Конвертируем радианы в градусы**

```
using System;

public static double ConvertRadiansToDegrees(double radians)
{
    double degrees = (180 / Math.PI) * radians;
    return degrees;
}
```

# Четное или нечетное число

Чтобы определить, является число четным или нечетным, достаточно разделить это число на два и посмотреть на его остаток. Если он равен 0, то число четное, если остаток равен 1, то число нечетное. Можно использовать следующие два метода (листинг 3.6) для решения этой задачи.

**Листинг 3.6. Определение четности числа**

```
public static bool IsEven(int intValue)
{
    return ((intValue % 2) == 0); // четное число
}
```

```
public static bool IsOdd(int intValue)
{
    return ((intValue % 2) == 1); // нечетное число
}
```

На самом деле не обязательно иметь два метода, а можно написать более универсальный код. Если число не является четным, значит, оно нечетно. Напишите этот метод сами. Если вам надо работать с другими типами целых чисел, то нужно добавить перегруженную версию метода (листинг 3.7).

**Листинг 3.7. Перегруженная версия метода `IsEven` для чисел типа `Long`**

```
// перегруженная версия, работающая с числами типа long
public static bool IsEven(long lValue)
{
    return ((lValue %2) == 0);
}
```

# Получить старшее и младшее слова из числа

Иногда требуется получить младшее (последние 16 бит) и старшее слово (первые 16 бит) из 32-битного целого числа. Для получения старшего слова из числа типа `int` нужно применить побитовое сравнение с помощью AND, как показано в листинге 3.8.

**Листинг 3.8. Получение старшего и младшего слов из 32-битного целого числа**

```
public static int GetHighWord(int intValue)
{
    return (intValue & (0xFFFF << 16));
}

public static int GetLowWord(int intValue)
{
    return (intValue & 0x0000FFFF);
}
```

# Преобразование градусов по Фаренгейту в градусы по Цельсию

Немецкий ученый Габриель Фаренгейт предложил свою систему для измерения температур. Существует несколько версий происхождения шкалы. По одной из них, Фаренгейт принял за 0 °F температуру около своего дома в один зимний день, а за 100 °F — температуру своего тела. Таким образом, диапазон градусов от 0° до +100° по шкале Фаренгейта примерно соответствует диапазону −18°…+38° по шкале Цельсия. Как видите, Фаренгейт ошибся в последнем измерении: нормальная температура человеческого тела составляет 97,9 °F. Данная шкала температур была широко распространена в англоязычных странах, но постепенно была вытеснена шкалой Цельсия. На нынешний момент эта система широко используется в США, Канаде и на Ямайке. Для перевода градусов по Фаренгейту в градусы по Цельсию существует специальная формула:

```
Celsius = (5 / 9) * (Fahrenheit - 32)
```

Используя данную формулу, можно без труда написать свой метод (листинг 3.9) для перевода градусов из одной меры в другую.

**Листинг 3.9. Конвертируем градусы Фаренгейта в градусы Цельсия**

```
public static double FtoC(double Fahrenheit)
{
    return ((5.0/9.0) * (Fahrenheit - 32));
}
```

Система градусов по Цельсию представляется более удобной. Вода замерзает при 0 градусов и закипает при 100 градусах. Таким образом, этой системой можно пользоваться везде, где есть вода. Для обратного конвертирования из градусов по Цельсию в градусы по Фаренгейту используется обратная формула. Напишем новый метод для этой цели (листинг 3.10).

**Листинг 3.10. Конвертируем градусы Цельсия в градусы Фаренгейта**

```
public static double CtoF(double Celsius)
{
    return (((9.0/5.0) * Celsius) + 32);
}
```

Существует еще одна шкала для измерения температуры — шкала Кельвина. Величина градуса Кельвина совпадает в величиной градуса Цельсия. Вся разница заключается в определении начальной точки отсчета. Начало шкалы (0 К) совпадает с абсолютным нулем (−273.15 градусов по Цельсию). Следовательно, для перевода температуры, выраженной в градусах по Кельвину, в систему по Цельсию достаточно от заданной величины отнять 273.15. Остается только добавить, что с 1968 года градус Кельвина официально носит собственное название — кельвин. Пример конвертации между градусами Цельсия и кельвинами мы приводить не будем из-за очевидной простоты.

# Генерирование случайного цвета

Метод `RandomRGBColor`, определенный в листинге 3.11, генерирует случайным образом красную, зеленую и синюю компоненты цвета.

**Листинг 3.11. Генерация компонентов цветов случайным образом**

```
private Random m_Rnd = new Random();

private Color RandomRGBColor()
{
    return Color.FromArgb(255, m_Rnd.Next(0,255),
        m_Rnd.Next(0,255), m_Rnd.Next(0,255));
}

private void button1_Click(object sender, EventArgs e)
{
    this.BackColor = RandomRGBColor();
}
```

# Подсчет суммы всех целых чисел диапазона

Хотите подсчитать сумму всех целых чисел в некотором диапазоне? Напишите соответствующую процедуру (листинг 3.12).

**Листинг 3.12. Подсчет суммы всех целых чисел диапазона**

```
// Подсчет суммы целых чисел из заданного диапазона
public static int SumAll(byte Value)
{
    int sum;
    sum = 0;

    for (int i = 1; i <= Value; i++)
        sum = sum + i;

    return sum;
}

Console.Write("Введите число от 1 до 125");
Console.WriteLine();
su = Convert.ToByte(Console.ReadLine());
Console.WriteLine(SumAll(su));
Console.ReadLine();
```

Заметим, что этот способ довольно неэффективен, и в реальных приложениях лучше использовать формулу для суммы арифметической прогрессии:

```
sum = Value * (Value + 1) % 2;
```

# Нахождение простых чисел

Простыми числами называют натуральные числа, большие единицы, которые делятся только на 1 и на себя. Примерами таких чисел являются 3, 5, 7. Если вы хотите получить список простых чисел из некоторого диапазона вручную, то у вас это займет очень продолжительное время. Поручим эту задачу компьютеру.

**Листинг 3.13. Получение списка простых чисел**

```
// нахождение простых чисел
static int[] GetSimpleNumbers(int Value)
{
    if (Value < 2)
    {
        int[] A = new int[0];
```

```
        return A;
    }
    else
    {
        int[] T = new int[Value];
        T[0] = 2;
        int K = 1, I = 3;
        bool B = true;
        while (I <= Value)
        {
            B = true;
            for (int J = 0; J < K; J++)
                if (I % T[J] == 0)
                {
                    B = false;
                    break;
                }
            if (B) T[K++] = I;
            I += 2;
        }
        int[] A = new int[K];
        for (int J = 0; J < A.Length; J++)
            A[J] = T[J];
        return A;
    }
}

// Выводим список простых чисел
Console.Write("Введите число от 2 до 100000");
Console.WriteLine();
x = Convert.ToInt32(Console.ReadLine());
int[] Array = GetSimpleNumbers(x);
//вывод чисел в консоль
for (int i = 0; i < Array.Length; i++)
    Console.Write("{0} ", Array[i]);
Console.ReadLine();
```

На сайте Википедии имеется страница "Список простых чисел" (найдите ее самостоятельно), в котором приводится список первых 500 простых чисел. Вы можете сравнить результаты работы программы с этим списком. Я попробовал вывести список простых чисел из диапазона от 2 до 1 000 000

(рис. 3.1). Обработка данных у меня заняла несколько секунд. Дальше я не решился экспериментировать.

Заметим, что этот алгоритм проверяет, является ли текущее нечетное число простым, сравнивая с нулем остаток от деления его на все простые числа, меньшие данного. Этот алгоритм можно существенно улучшить, если принять во внимание, что на самом деле достаточно брать лишь простые числа, не превосходящие корня из заданного числа. Ведь если число делится на какое-нибудь простое, большее корня из него, то оно делится и на частное, которое окажется меньше корня.

### *Примечание*

Рассмотренные выше примеры находятся в папке Algorithm на прилагаемом диске.

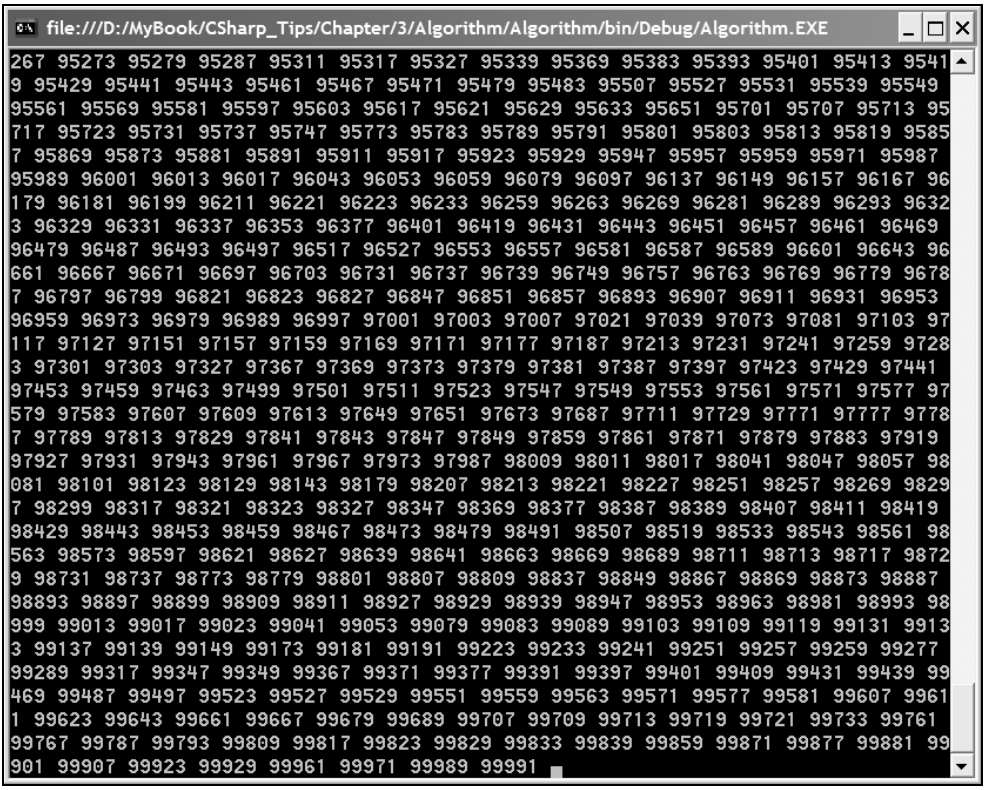



**Рис. 3.1.** Список простых чисел

# Вывод программой своего исходного кода

Забавы ради многие программисты пишут программы, которые при запуске выводят содержимое своего исходного кода. Такие программы существуют практически на всех языках программирования и носят специальное название куайн (quine). В Википедии вы можете почитать более подробно на этот счет. Там же приводится и реализация на C#, представленная в листинге 3.14.

**Листинг 3.14. Пример программы, выводящей свой код**

```
using System;
class A
{
    static void Main()
    {
        string s = "using System;class A{{static void Main(){{string
s={0}{1}{0};char q='{0}';Console.Write(s,q,s);}}}}";
        char q = '"'; Console.Write(s,q,s);
    }
}
```

В данном листинге переносы строк использованы для удобства, на самом деле их быть не должно. Вы можете убрать эти переносы и запустить пример. Результат представлен на рис. 3.2.

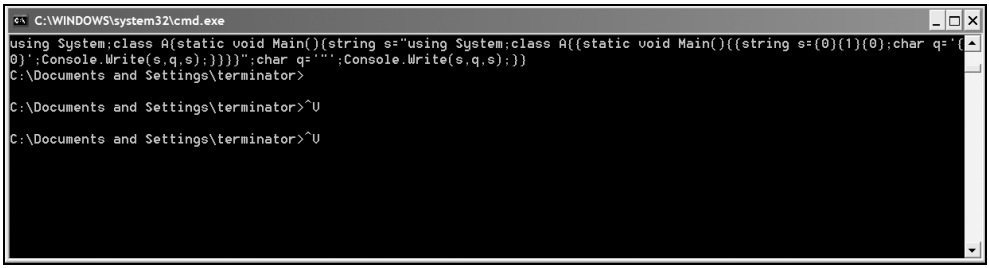



**Рис. 3.2.** Программа, которая выводит свой код на экран

Сергей Борзов в своем блоге приводит еще две ссылки на программы-куайны, а также предлагает свой вариант. Почитайте его пост по адресу

**http://seregaborzov.wordpress.com/2007/06/09/csharp-quine-and-pascal-triangle/**. Там же вы найдете алгоритм создания треугольника Паскаля.

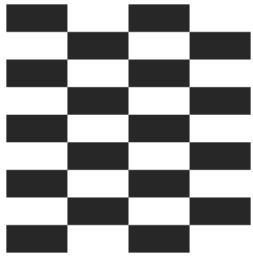
***Примечание***

Пример создания куайна находится в папке Quine на прилагаемом диске.

# Заключение

Простые алгоритмы, рассмотренные в этой главе, помогут вам в ряде случаев. Однако в реальных приложениях часто приходится сталкиваться с гораздо более сложными алгоритмами.

ГЛАВА 4



# Интегрированная среда разработки (IDE)

С выходом каждой новой версии Visual Studio среда разработки становится все лучше, удобнее и сложнее. Интегрированная среда разработки Visual Studio 2008 является очень мощным пакетом для создания сложных программ на любой вкус. Причем она настолько мощна, что многие программисты даже не в курсе многих ее возможностей и используют только малую часть ее функциональности. Не случайно имеются даже целые книги, целиком посвященные описанию работы с интегрированной средой разработки. По адресу **http://msdn2.microsoft.com/en-us/library/bb245788(vs.80).aspx** имеется небольшая подборка советов по настройке IDE на английском языке. Хочу предложить вам часть полезных советов из этой статьи, а также несколько собственных трюков.

## Удобные клавиатурные команды

Если вы опытный пользователь, то используйте во время работы назначенные клавиши для более быстрого выполнения различных операций. Некоторые из них вы наверняка уже знаете: <F5> для команды `Debug.Start`, <F10> для команды `Debug.StepOver`, <F4> для команды `View.Properties`. В табл. 4.1 приведены некоторые другие, менее известные назначенные клавиши.

***Таблица 4.1.*** *Комбинации клавиш*

| Назначенная клавиша | Команда |
|---|---|
| <F7> | Переключение между режимами дизайна формы и редактора кода |

***Таблица 4.1*** *(окончание)*

| Назначенная клавиша | Команда |
|---|---|
| <F9> | Добавление/снятие точки останова |
| <F12> | Переход к определению переменной, объекта или функции |
| <Ctrl>+<Shift>+<7> и <Ctrl>+<Shift>+<8> | Быстрое перемещение вперед и назад в стеке определений |
| <Shift>+<F12> | Поиск всех ссылок на функцию или определение |
| <Ctrl>+<M>, <Ctrl>+<M> | Развертывание и свертывание структуры кода в редакторе |
| <Ctrl>+<K>, <Ctrl>+<C> и <Ctrl>+<K>, <Ctrl>+<U> | Добавление или удаление знака комментария к строке(-ам) кода соответственно |
| <Alt>+<Shift>+<Enter> | Переключение между полноэкранным и обычным режимами |
| <Ctrl>+<I> | Последовательный поиск |

# Получение списка всех назначенных клавиш

Оказывается, в среде разработки Visual Studio 2005/2008 имеется более 450 назначенных клавиш. Понятно, что запомнить или найти их все нелегко. Но можно написать специальный макрос, который поможет получить их значения по умолчанию. Чтобы написать такой макрос, необходимо в меню **Tools** выбрать команды **Macros** | **Macros IDE**… У вас запустится специальная среда разработки для создания макросов. Слева в иерархическом списке разверните проект **MyMacros** и дважды щелкните на модуле **Module1**. Откроется редактор кода, в котором нужно добавить текст программы, генерирующей HTML-страницу. В этой странице будет находиться таблица всех используемых назначенных клавиш. Код этого макроса приведен в листинге 4.1.

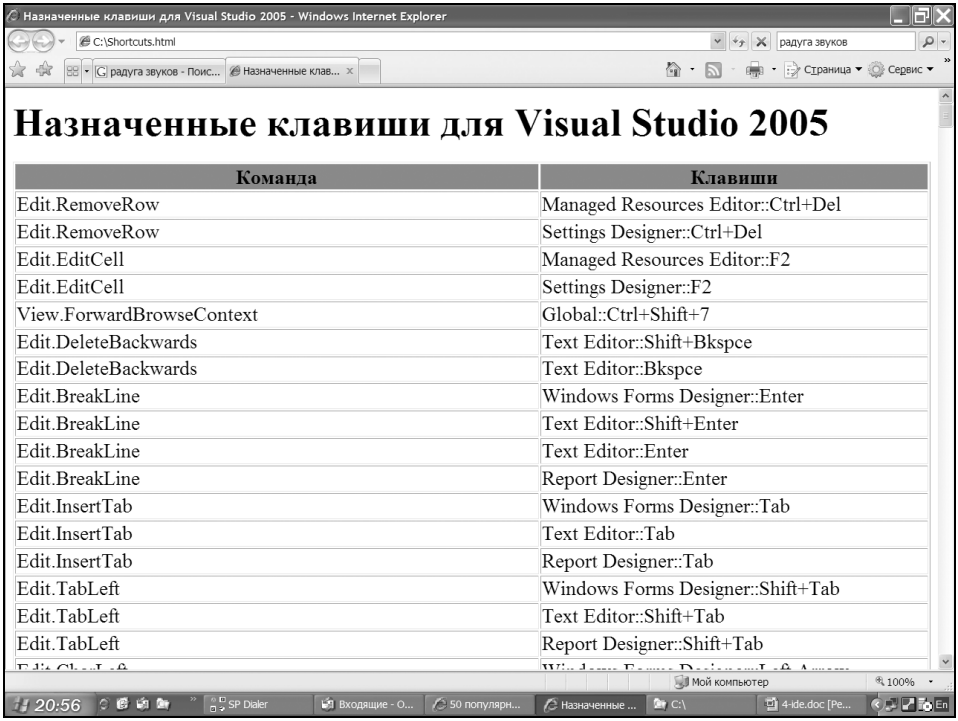



**Рис. 4.1.** Фрагмент списка назначенных клавиш Visual Studio 2005

**Листинг 4.1. Макрос для создания списка назначенных клавиш в формате HTML**

```
Imports System
Imports EnvDTE
Imports EnvDTE80
Imports System.Diagnostics

Public Module Module1
    Public Sub ListShortcutsInHTML()

        ' Объявляем StreamWriter и определяем имя файла справки
        Dim sw As System.IO.StreamWriter
        sw = New System.IO.StreamWriter("c:\\Shortcuts.html")

        ' Записываем начало html-страницы
        WriteHTMLStart(sw)
```

```
    ' Добавляем ряд для каждой комбинации клавиш
    For Each c As Command In DTE.Commands
        If c.Name <> "" Then
            Dim bindings As System.Array
            bindings = CType(c.Bindings, System.Array)
            For i As Integer = 0 To bindings.Length - 1
                sw.WriteLine("<tr>")
                sw.WriteLine("<td>" + c.Name + "</td>")
                sw.WriteLine("<td>" + bindings(i) + "</td>")
                sw.WriteLine("</tr>")
            Next
        End If
    Next

    ' Конец html-страницы
    WriteHTMLEnd(sw)

    ' Очищаем буфер потока и закрываем его
    sw.Flush()
    sw.Close()
End Sub

' Пишем начальный код для страницы: шапку, заголовок и т.д.
Public Sub WriteHTMLStart(ByVal sw As System.IO.StreamWriter)
    sw.WriteLine("<html>")
    sw.WriteLine("<head>")
    sw.WriteLine("<meta http-equiv=Content-Type " + _
                 "content=""text/html; charset=UTF-8"">")
    sw.WriteLine("<title>")

    sw.WriteLine("Назначенные клавиши для Visual Studio 2005")
    sw.WriteLine("</title>")
    sw.WriteLine("</head>")

    sw.WriteLine("<body>")
    sw.WriteLine("<h1>Назначенные клавиши для " + _
                 "Visual Studio 2005</h1>")
    sw.WriteLine("<table border=""1"">")
    sw.WriteLine("<tr BGCOLOR=""#018FFF"" align=""center"">" + _
                 "<th>Команда</th><th>Клавиши</th></tr>")

End Sub
```

```
    Public Sub WriteHTMLEnd(ByVal sw As System.IO.StreamWriter)
        sw.WriteLine("</table>")
        sw.WriteLine("</body>")
        sw.WriteLine("</html>")
    End Sub
End Module
```

Запустите написанный макрос. В результате его выполнения на диске C: будет создан файл Shortcuts.html. Этот файл, открытый в браузере, показан на рис. 4.1. Вы можете открывать его в любое время, или же распечатать и повесить на стенку рядом с компьютером.

## Настройка назначенных клавиш

Сколько людей, столько и мнений. Возможно, у вас уже есть любимые назначенные клавиши, которые не совпадают с назначенными по умолчанию. Чтобы не изменять своим привычкам, вы можете настроить назначенные клавиши под себя. Для этого выберите в меню команду **Tools | Options...** и в открывшемся окне выберите страницу **Environment | Keyboard** (рис. 4.2).

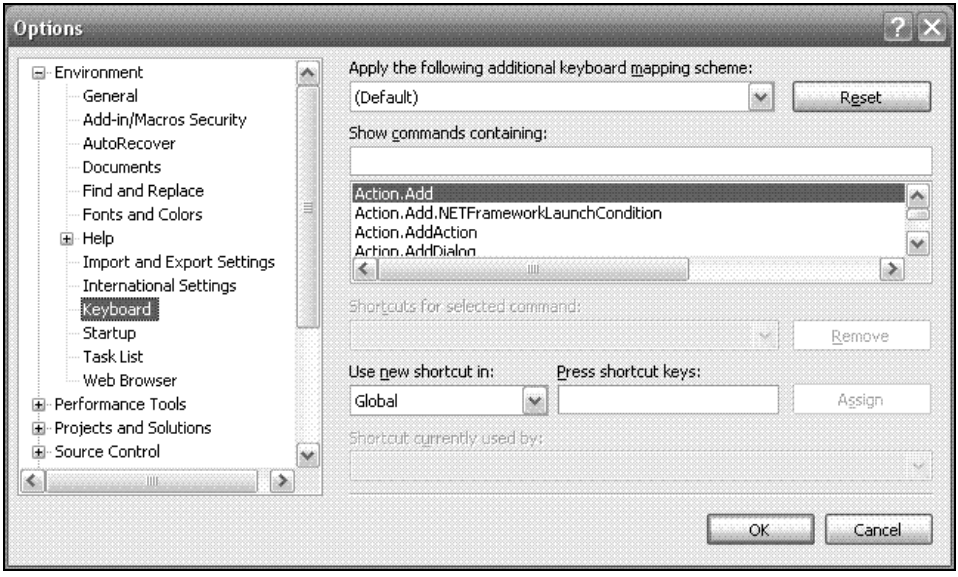



**Рис. 4.2.** Диалоговое окно параметров — настройка назначенных клавиш

Однако существует другой, более простой способ, чем добавлять множество новых назначенных клавиш в окне настроек программы. Список назначенных клавиш проще изменить, непосредственно исправив файл, в котором автоматически сохраняются параметры. Делается это следующим образом.

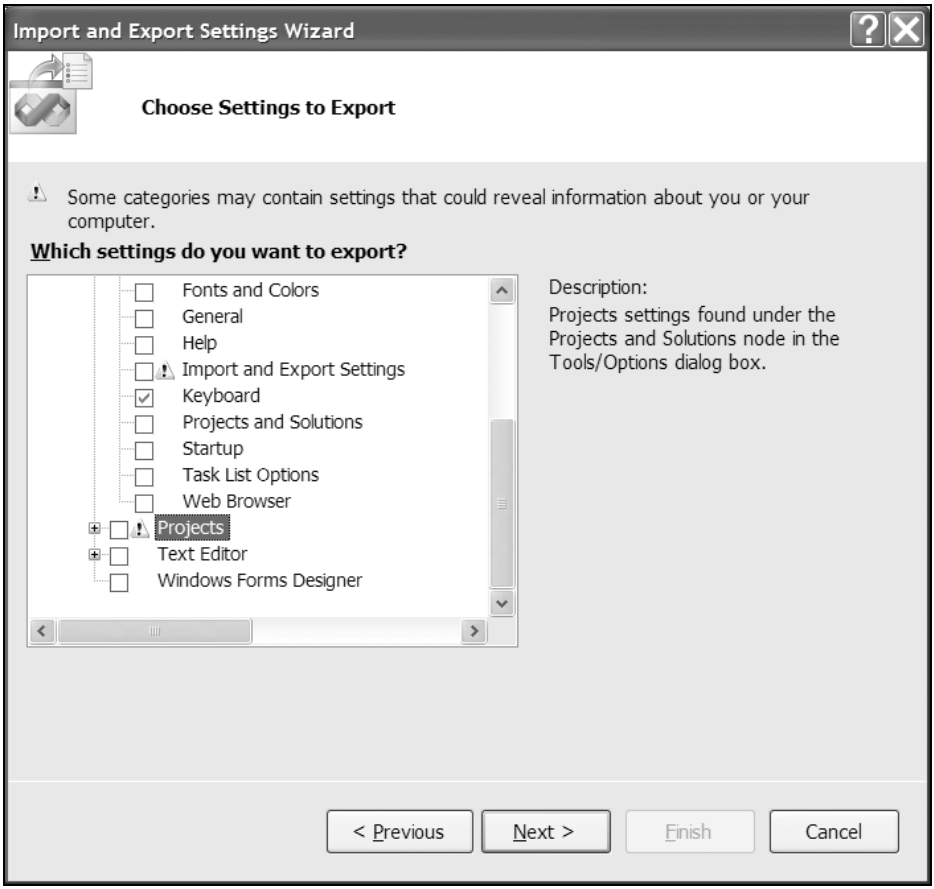



**Рис. 4.3.** Выбор только параметров клавиатуры для экспорта

Сначала экспортируем текущий список назначенных клавиш. Для этого выберите меню **Tools | Import and Export Settings...**. Откроется окно мастера импорта/экспорта параметров. Отметьте вариант **Export selected environment settings** (Экспорт выбранных настроек окружения) и нажмите на кнопку **Next**. Далее выберите **All Settings**, чтобы снять все флажки, затем разверните секции **Options** и **Environment** и поставьте флажок в ячейке

**Keyboard**. Далее нажмите кнопку **Next**, чтобы перейти к последней странице мастера. Выберите имя для создаваемого файла параметров, например, MyKeyboardShorcuts.vssettings, и оставьте путь к каталогу по умолчанию. Для завершения работы мастера экспорта нажмите кнопку **Finish**.

Теперь вы можете отредактировать файл параметров. Найдите созданный файл (по умолчанию он будет находиться по адресу My Documents\Visual Studio 2008\Settings\MyKeyboardShortcuts.vssettings). Настройки в файле параметров записаны в формате XML, и их можно открыть в любом текстовом редакторе. Можно открыть файл в самой программе Visual Studio 2008. В этом случае редактор кода выделит синтаксис цветом и отформатирует документ, что намного облегчает работу с XML. После открытия файла нажмите комбинацию клавиш <Ctrl>+<K>, <Ctrl>+<D>, и текст будет автоматически отформатирован Visual Studio. Затем найдите тег `<UserShortcuts>`. В этот XML-элемент можно добавить собственный список назначенных клавиш. В листинге 4.2 приводится небольшая часть файла.

**Листинг 4.2. Добавление назначенных клавиш непосредственно в файл параметров.**

```
...
<UserShortcuts>
   <Shortcut Command="View.CommandWindow" Scope="Global">
Ctrl+W, Ctrl+C
</Shortcut>
   <Shortcut Command="View.SolutionExplorer" Scope="Global">
Ctrl+W, Ctrl+S
</Shortcut>
   <Shortcut Command="View.ErrorList" Scope="Global">
Ctrl+W, Ctrl+E
</Shortcut>
   <Shortcut Command="View.TaskList" Scope="Global">
Ctrl+W, Ctrl+T
</Shortcut>
   <Shortcut Command="View.Output" Scope="Global">
Ctrl+W, Ctrl+O
</Shortcut>
</UserShortcuts>
...
```

Здесь формат XML достаточно прост. Каждой добавляемой комбинации клавиш соответствует элемент `<Shortcut>`. Назначенная клавиша указывается

как содержимое элемента, модификаторы `Shift`, `Ctrl`, `Alt` используются вместе с символом `+` (например, `Ctrl+Alt+J`). Атрибут `Scope` (область действия) почти всегда равен `Global` (глобально), поэтому мы не будем рассматривать его подробно. В атрибуте `Command` для команды, для которой необходимо назначить клавишу, указывается стандартное имя команды. Некоторую трудность может составить выяснение имени для определенной команды. Стандартное имя команды состоит из имени меню верхнего уровня и имени команды, разделяемых точкой. После добавления всех назначенных клавиш сохраните файл.

Далее ваш отредактированный файл нужно импортировать обратно в среду разработки. Естественно, созданный файл параметров могут использовать и другие пользователи. Снова запустите мастер импорта и экспорта параметров и выберите параметр **Import selected environment settings** (Импорт выбранных настроек окружения). Далее нажмите кнопку **Next**, выберите команду **No, just import new settings, overwriting my current settings** (Нет, лишь импортировать новые настройки, перезаписывая текущие) и снова нажмите кнопку **Next**. Выберите ваш файл MyKeyboardShortcuts.vssettings в папке My Settings и нажмите кнопку **Next**. Оставьте настройки по умолчанию и нажмите кнопку **Finish**. Если вы все сделали правильно, то ваши новые настройки будут успешно импортированы в Visual Studio 2005.

## Показ назначенных клавиш во всплывающих подсказках

Можно настроить среду разработки таким образом, чтобы IDE показывала назначенные клавиши во всплывающих подсказках, появляющихся при наведении курсора мыши на команды на панелях инструментов. Откройте меню **Tools | Customize...** и убедитесь, что установлен флажок **Show shortcut keys in ScreenTips** (Показывать назначенные клавиши во всплывающих подсказках), показанный на рис. 4.4.

## Селектор оконных конфигураций

Среда разработки Visual Studio 2008 является очень гибким инструментом, который позволяет настраивать среду разработки под различные вкусы. Рассмотрим, как можно быстро переключаться между различными макетами окон в соответствии с текущей задачей.

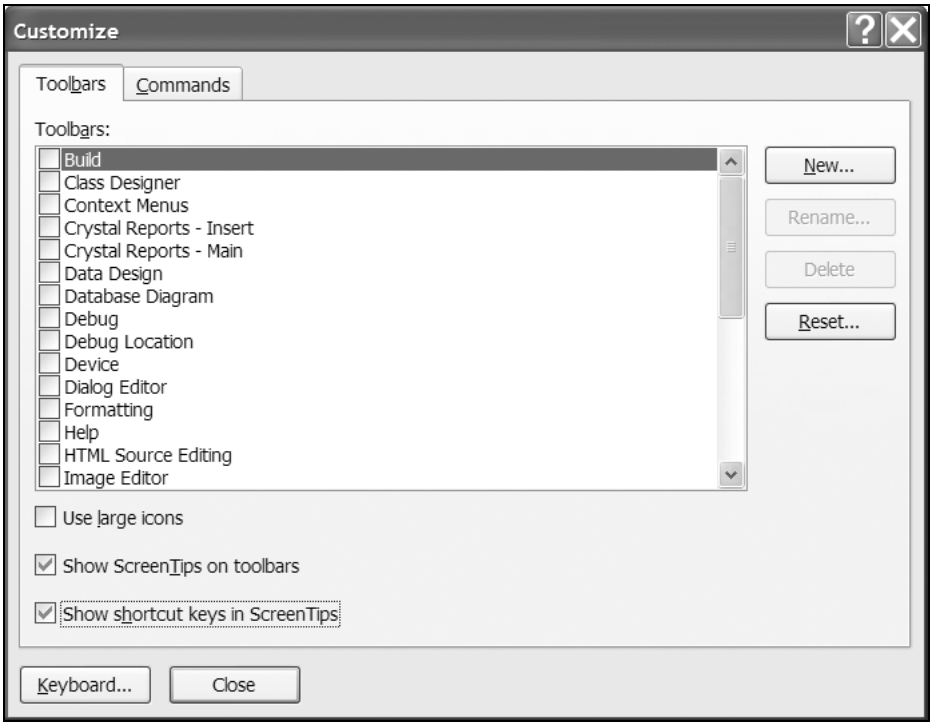



**Рис. 4.4.** Включение показа назначенных клавиш во всплывающих подсказках

Мы уже говорили о способах импорта или экспорта параметров среды, касающихся назначенных клавиш. Но на самом деле можно экспортировать в файл практически все настройки среды, который затем может использоваться другими пользователями, или сохранить этот файл в качестве резервной копии. В число параметров, которые можно импортировать или экспортировать, входят конфигурация окон, назначенные клавиши, настройки меню, шрифты и цвета и другие настройки из меню **Tools** | **Options…** При создании макроса для выбора конфигурации окон сначала нужно создать отдельный файл параметров для каждой конфигурации. В качестве примера мы создадим три файла параметров, соответствующие трем расположениям окон: CodeWriting, CodeBrowsing и FormsDesign.

Расположите и настройте поведение окон, чтобы оно было удобно при написании кода. Большинство программистов предпочитают устанавливать для всех видимых окон инструментов автоматическое свертывание, чтобы максимально увеличить пространство для написания кода. Затем откройте меню

**Tools | Import and Export Settings…**, чтобы запустить мастер импорта и экспорта параметров. Выберите **Export selected environment settings** и нажмите кнопку **Next**. Установите флажок у **Window Layouts** и нажмите кнопку **Next**. Назовите файл параметров CodeWritingWinLayout.vssettings и нажмите кнопку **Finish**. Таким образом вы создали первый из трех необходимых файлов параметров. Повторите указанные выше действия для двух оставшихся файлов параметров. Очевидно, необходимо изменить макет окна и соответственно назвать файлы, например CodeBrowsingWinLayout.vssettings и FormsDesignWinLayout.vssettings.

## Создание макросов для импорта файлов параметров

После того как мы создали файл с параметрами, необходимо написать три макроса для импорта всех трех файлов. Код макросов приведен в листинге 4.3.

**Листинг 4.3. Код макроса для импорта файла параметров**

```
Imports EnvDTE
Imports EnvDTE80
Imports System.Diagnostics
Imports System.IO

Public Module Module1

  Public Sub ImportWinLayoutCodeWriting()
    DTE.ExecuteCommand("Tools.ImportandExportSettings",
    "-import:c:\demo\settings\CodeWritingWinLayout.vssettings")
  End Sub

  Public Sub ImportWinLayoutCodeBrowsing()
    DTE.ExecuteCommand("Tools.ImportandExportSettings",
    "-import:c:\demo\settings\CodeBrowsingWinLayout.vssettings")
  End Sub

  Public Sub ImportWinLayoutFormsDesign()
    DTE.ExecuteCommand("Tools.ImportandExportSettings",
    "-import:c:\demo\settings\FormsDesignWinLayout.vssettings")
End Sub

End Module
```

### Добавление кнопок на панель инструментов

Теперь необходимо создать кнопки на панели инструментов для изменения текущей конфигурации окна. Выберите команды **Tools** | **Customize…** в меню и вкладку **Commands** в открывшемся окне. В списке **Categories** выберите **Macros**, затем в списке **Commands** найдите три только что созданных макроса: `MyMacros.Module1.ImportWinLayoutCodeWriting`, `MyMacros.Module1.ImportWinLayoutCodeBrowsing` и `MyMacros.Module1.ImportWinLayoutFormsDesign`.

Перетащите эти команды на панель инструментов Visual Studio. Теперь можно щелкнуть правой кнопкой мыши на кнопках на панели инструментов и дать им более короткие имена.

Закройте диалоговое окно **Customize**, при этом сделанные настройки будут сохранены. Создание селектора макетов окна завершено. Нажмите новые кнопки на панели инструментов, чтобы проверить их работу. Этим командам можно даже назначить клавиши в окне, отрываемом командой меню **Tools** | **Options…** на странице **Environment** | **Keyboard**.

## Фрагменты кода (code snippets)

Code snippets (фрагменты кода) — одна из лучших и наиболее удобных функций, которая появилась еще в Visual Studio 2005. Фрагменты кода повышают производительность труда программиста, избавляя его от рутинной работы. С помощью заранее подготовленных фрагментов кода можно быстро вставлять однотипные куски кода (например, код для цикла `for`). Также имеются шаблоны для типовых задач (отправка данных в сети и т. д.). Большинство предлагаемых в Visual Studio 2008 фрагментов C# относятся к первому типу — они помогают минимизировать ввод повторяющегося кода. А в Visual Basic больше фрагментов второго типа — они помогут быстрее программировать определенные задачи.

Существуют два способа вставки фрагментов. Можно ввести название (alias) фрагмента кода в редакторе и два раза нажать клавишу <Tab>. Например, введите в редакторе кода слово `for` и дважды нажмите клавишу <Tab>. У вас сразу появится готовая заготовка для этого цикла. Осталось только подкорректировать некоторые переменные. После вставки фрагмента кода для перехода к различным полям во фрагменте используйте клавиши <Tab> и <Shift>+<Tab>. Это позволит быстро изменить необходимые части кода. Об-

ратите внимание, что в C# названия фрагментов кода видны в IntelliSense. В списке IntelliSense их можно определить по специальному значку (рис. 4.5).

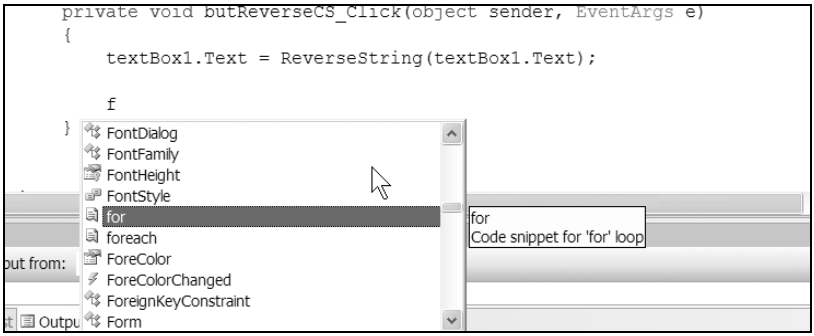



**Рис. 4.5.** Подсказка IntelliSense работает с фрагментами кода

Интересная особенность — фрагменты кода обладают некоторой интеллектуальностью. Приведу абстрактный пример для демонстрации. Предположим, вы объявили некоторое перечисление `WeekDayName` (листинг 4.4).

**Листинг 4.4. Объявление перечисления**

```
public enum WeekDayName
{
    Понедельник = 1,
    Вторник = 2,
    Среда = 3,
    Четверг = 4,
    Пятница = 5,
    Суббота = 6,
    Воскресенье = 7
}
```

Далее, вы создаете свою процедуру, в которой будет использоваться оператор `switch`. Итак, сначала мы пишем заготовку для процедуры, то есть имя процедуры с круглыми скобками, и пару фигурных скобок, в которых располагается тело процедуры. Теперь внимание. Начинаем вводить первую букву `s`, IntelliSence предлагает на выбор несколько слов, из которых мы выбираем `switch` и нажимаем на клавишу <Tab>. Далее сразу еще раз нажимаем на

<Tab>, чтобы воспользоваться готовым фрагментом кода. У нас появится что-то вроде кода, приведенного в листинге 4.5.

**Листинг 4.5. Формирование новой процедуры**

```
private void InfoAboutWeekDay(WeekDayName wd)
{
    switch (switch_on)
    {
        default:
    }
}
```

Обратите внимание, что слово `switch_on` выделено зеленым прямоугольником. Введите в этом месте слово `wd` и нажмите клавишу <Enter>. Редактор кода автоматически пройдется по всем элементам перечисления и создаст подходящий код, который будет выглядеть, как в листинге 4.6.

**Листинг 4.6. Автоматическое заполнение оператора `switch`**

```
switch (wd)
{
    case WeekDayName.Понедельник:
        break;
    case WeekDayName.Вторник:
        break;
    case WeekDayName.Среда:
        break;
    case WeekDayName.Четверг:
        break;
    case WeekDayName.Пятница:
        break;
    case WeekDayName.Суббота:
        break;
    case WeekDayName.Воскресенье:
        break;
    default:
        break;
}
```

Всего за несколько секунд и пару нажатий клавиш вы получили готовый код оператора `switch`, в который осталось лишь добавить функциональную часть. При большом количестве элементов перечисления подобный прием позволяет сэкономить много времени.

Если вы не помните название фрагмента, его можно выбрать из списка. Для тех, кто привык работать с мышью, нужно щелкнуть правой кнопкой мыши и выбрать команду **Insert Snippet...**. На экране появится окно выбора, в котором можно увидеть все фрагменты кода для данного языка и выбрать необходимый фрагмент (рис. 4.6). Те, кто предпочитает работать с клавиатурой, могут вызвать этот список с помощью комбинации клавиш. Для этого сначала нужно нажать в редакторе кода комбинацию <Ctrl>+<K>, после чего в нижней части программы появится сообщение **(Ctrl+K) was pressed. Waiting for second key of chord…** (Было нажато Ctrl+K, жду вторую клавишу сочетания). Теперь нажмите комбинацию клавиш <Ctrl>+<X>, и у вас появится этот же список, показанный на рис. 4.6.

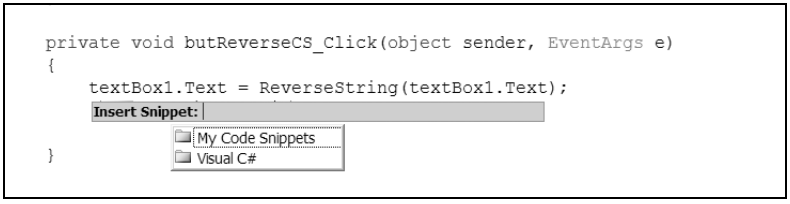



**Рис. 4.6.** Вставка фрагмента кода в C#

Но на самом деле можно использовать не только предопределенные фрагменты кода, но и самому создавать собственные code snippets и загружать фрагменты, созданные другими разработчиками.

Создать собственные фрагменты кода в Visual Studio очень просто. Рассмотрим случай на конкретном примере. Вероятно, вам часто приходится использовать одни и те же операции при написании кода: открыть файл, выполнить какую-либо обработку, затем закрыть файл. Ниже показано создание такого фрагмента кода.

## Создание XML-файла

Фрагмент кода описывается в отдельном файле в формате XML. В Visual Studio откройте меню **File | New… | File...** и выберите тип файла XML (рис. 4.7).

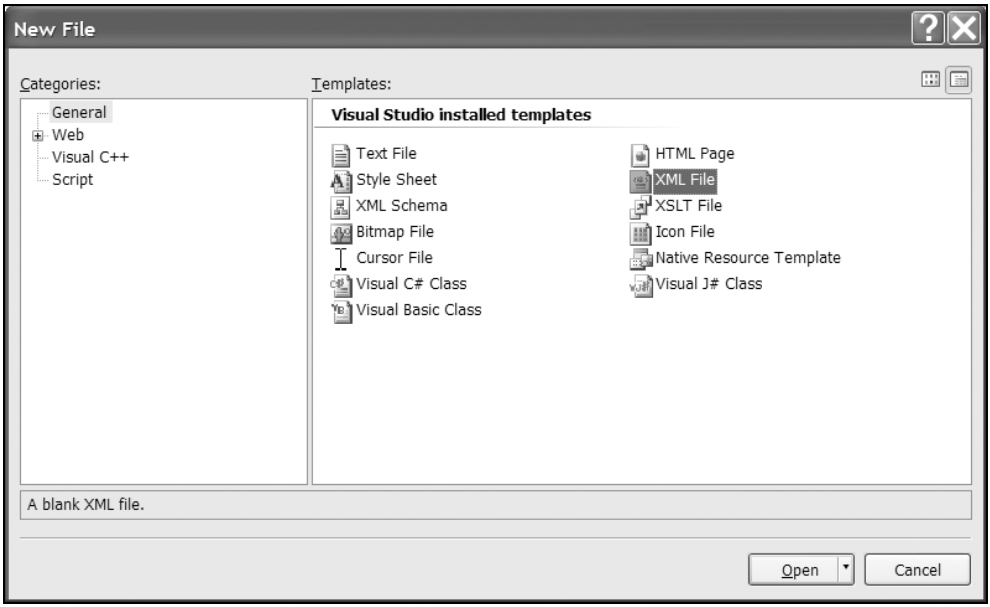



**Рис. 4.7.** Создание нового XML-файла

Что интересно, для создания фрагмента кода существует собственный специальный фрагмент кода. На второй строке файла нажмите <Ctrl>+<K>, <Ctrl>+<X>, выберите пункт **Snippet** (рис. 4.8).

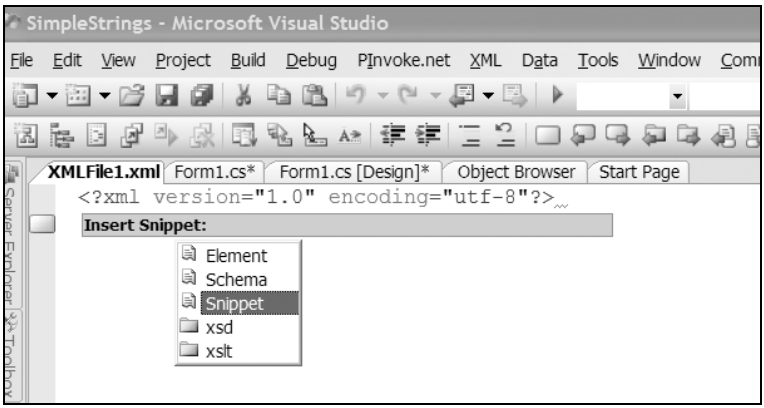



**Рис. 4.8.** Использование XML-фрагмента для создания других фрагментов кода

Когда вы выберете эту команду, то шаблон файла фрагмента кода будет вставлен автоматически в редактор кода (рис. 4.9).


**Рис. 4.9.** Заготовка для фрагмента кода

Поля `Title` (название фрагмента кода), `Author` (автор фрагмента), `Shortcut` (ярлык) и `Description` (описание) говорят сами за себя, поэтому не будем подробно на них останавливаться. Содержимое тегов заслуживает отдельного разговора, поэтому разберем примеры.

Основной код находится в теге `<![CDATA[...]]>` (листинг 4.7), который, в свою очередь, находится внутри тега `<Code>`. Поля, которые должны меняться пользователями, необходимо окружить символами доллара (`$`). В нашем примере для свободной замены пользователями фрагмента кода предусмотрены три текстовых поля (литерала): `StrmReader`, `FilePath` и `Line`. Они находятся в разделе `CDATA` и окружены знаками `$`. Кроме того, все текстовые константы должны быть определены в элементе `<Declarations>`. Каждой из них дается идентификатор и дополнительное значение по умолчанию.

Кроме того, в коде имеется литерал, который не был определен: `$end$`. Это специальный литерал, указывающий расположение курсора после нажатия пользователем клавиши <Enter>, когда все поля фрагмента кода заполнены. Существует еще один специальный литерал, который здесь не используется: `$selected$`. Он применяется только для фрагментов кода типа `SurroundsWith`. Данный литерал определяет место размещения выбранного фрагмента кода при вставке этого фрагмента с помощью команды **Surround With...**.

**Листинг 4.7. Пример фрагмента кода**

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippet Format="1.0.0"
xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <Header>
    <Title>File Processing</Title>
    <Author>James Lau</Author>
    <Shortcut>fp</Shortcut>
    <Description>Opens a file, does some processing,
      and then closes the file.</Description>
    <SnippetTypes>
      <SnippetType>SurroundsWith</SnippetType>
      <SnippetType>Expansion</SnippetType>
    </SnippetTypes>
  </Header>
  <Snippet>
    <Declarations>
      <Literal>
        <ID>StrmReader</ID>
        <Default>strmReader</Default>
      </Literal>
      <Literal>
        <ID>FilePath</ID>
        <Default>fPath</Default>
      </Literal>
      <Literal>
        <ID>Line</ID>
        <Default>strLine</Default>
      </Literal>
    </Declarations>
    <Code Language="CSharp">
```

```
      <![CDATA[
   StreamReader $StrmReader$ = null;
   try
   {
      $StrmReader$ = new StreamReader($FilePath$);
      string $Line$;
      while (($Line$ = $StrmReader$.ReadLine()) != null)
      {
         // Perform some processing
         $selected$
         $end$
      }
   }
   catch (IOException ioex)
   {
      // Handle exception
   }
   finally
   {
      $StrmReader$.Close();
   }
    ]]>
    </Code>
  </Snippet>
</CodeSnippet>
```

## Встроенные фрагменты кода

Visual Studio 2008 имеет в своем составе множество встроенных фрагментов кода. Найти относящиеся к C# фрагменты можно в папке C:\Program Files\Microsoft Visual Studio 9.0\VC#\Snippets\1033.

## Распространение собственных фрагментов кода

Если вы написали свой полезный фрагмент кода и хотите поделиться с человечеством, то просто найдите свой файл с расширением snippet, заархивируйте его архиватором типа WinZip или WinRAR и выложите его на своем сайте. Теперь любой желающий может скачать вашу разработку и интегрировать ее в свой пакет Visual Studio 2008. Для этой цели нужно воспользоваться менеджером фрагментов кода (Code Snippets Manager). Чтобы открыть менед-

жер фрагментов кода, нужно воспользоваться командами меню **Tools | Code Snippets Manager**. С помощью кнопки **Add** в диалоговом окне вы можете добавить целую папку, содержащую коллекцию фрагментов кода. Кнопка **Import** позволит добавить отдельный экземпляр фрагмента кода.

## Настройка стартовой страницы Visual Studio

Вы, наверное, обратили внимание, что на стартовой странице Visual Studio 2008 отображается RSS-канал, предоставляющий последние сведения о новостях MSDN. Вы можете настроить другой RSS-канал по своему желанию. Для этого в окне, открываемом командой меню **Tools | Options…**, на странице **Environment | Startup** нужно изменить URL-адрес в области **Start Page news channel** (Новостные каналы стартовой страницы). По умолчанию там стоит строка **http://go.microsoft.com/fwlink/?linkid=35587&clcid=409**. Также возможно определить частоту обновления новостей.

Если же вы не нуждаетесь в автоматическом отображении стартовой страницы при запуске Visual Studio, то в области **At startup** (при запуске) на этой же странице параметров можно выбрать пункт **Show empty environment** (Показывать пустое окружение).

## Настройки для групповой работы

Если программисты-любители работают в одиночку, то в крупных компаниях разработчики работают командой, где каждый отвечает за свой фронт работ. В этом случае полезными могут оказаться параметры Team Settings. При совместной работе эти параметры помогают в установлении правил программирования группы или в более быстрой настройке Visual Studio.

Предположим, необходимо настроить набор групповых правил форматирования кода. Чтобы не настраивать параметры IDE у всех членов группы в соответствии с этими правилами, можно просто создать файл параметров и предоставить его всем. При обновлении файла параметров группы он будет автоматически импортирован при очередном запуске Visual Studio поверх текущих параметров пользователя. Ниже приводится пример использования этой функции.

## Создание файла параметров

Функция Team Settings может использоваться для настройки любых параметров IDE, включая настройки шрифтов, цветов, назначенных клавиш, настроек меню. Как правило, особенно часто она используется разработчиками для настройки параметров форматирования кода. Необходимо лишь настроить необходимые параметры, а затем экспортировать параметры в файл с помощью известного нам способа (меню **Tools** | **Import/Export Settings…**). Нужно проследить, чтобы экспортировался только тот набор параметров, который нужен всей группе.

## Размещение файла параметров в пути UNC

Скопируйте файл параметров, созданный в предыдущем разделе, в сетевую папку, к которой имеют доступ все члены группы. Теперь члены группы должны задать путь к файлу настроек.

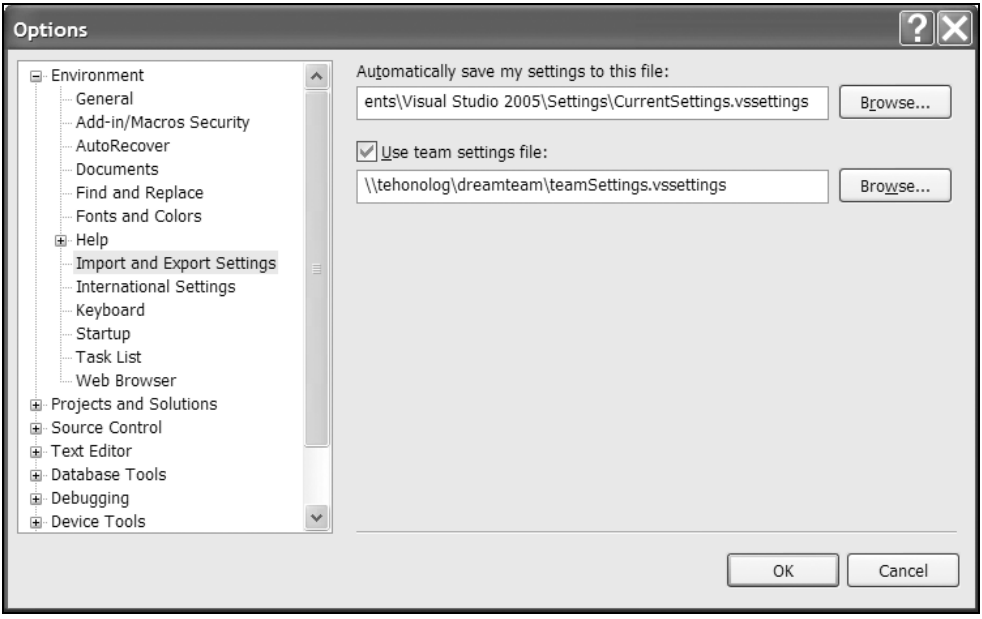



**Рис. 4.10.** Диалоговое окно **Options** для изменения пути параметров группы

Для этого нужно выбрать команду меню **Tools** | **Options…**, в открывшемся окне выбрать страницу **Environment** | **Import and Export Settings**, установить флажок **Use team settings file** (Использовать файл параметров группы) и указать путь к файлу настроек. На рис. 4.10 показаны настройки, когда файл параметров группы расположен по адресу \\tehnolog\dreamteam\teamsettings.settings.

## Параметр */resetuserdata*

Параметр `/resetuserdata` используется для восстановления исходного состояния Visual Studio в случае его повреждения и невозможности восстановления. Например, на вашем компьютере оказались поврежденными файлы конфигурации окон, настроек меню или назначенных клавиш. Использовать данный параметр нужно следующим образом. Закройте все экземпляры Visual Studio 2008. Нажмите кнопку **Пуск** и выберите команду **Выполнить...**. Введите в диалоговом окне команду:

```
devenv.exe /resetuserdata
```

Выполнение команды займет несколько минут, так как выполняется очистка Visual Studio и восстановление исходного состояния. Для проверки состояния процесса devenv.exe можно использовать Диспетчер задач. После завершения процесса необходимо перезапустить Visual Studio. У вас запустится диалоговое окно, которое появляется при первом запуске пакета Visual Studio на компьютере.

> ***ПРЕДУПРЕЖДЕНИЕ***
>
> При использовании этого параметра будут потеряны все настройки и параметры среды. Именно по этой причине этот параметр не документирован официально, и Microsoft не сообщает о нем пользователям. Его следует использовать только в качестве крайней меры при неполадках среды. Также желательно сделать резервную копию настроек путем экспорта в файл.

## Ряд мелких советов

Приведем теперь несколько коротких советов об использовании некоторых не очень хорошо известных функций среды разработки.

## Как показывать нумерацию строк в редакторе кода?

Выберите в меню пункт **Tools** | **Options…** и в открывшемся окне выберите страницу **Text Editor** | **All Languages** поставьте галочку **Line numbers**. После этого в редакторе будут пронумерованы строки. Можно также показывать нумерацию строк кода в зависимости от языка, выбирая страницу, соответствующую нужному языку в разделе **Text Editor**, например, страницу C# или Basic, и отмечая эту же опцию. Нумерацию строк удобно использовать в учебных материалах: можно указать на номер строки и объяснить, что делает код в указанной строке.

## Как изменить цвет для регионов кода?

На странице **Environment** | **Fonts and Colors** окна, открываемого командой меню **Tools** | **Options…**, нужно выбрать в списке **Display items** пункт **Collapsible Text**. Теперь можно установить цвет для регионов. На этой же странице можно настраивать шрифты и цвета для других участков кода.

## Вертикальное выделение текста

В редакторе кода текст выделять можно не только привычным способом, строчку за строчкой, но и колонками. Нажмите и удерживайте клавишу <Alt>, а мышью выделяйте текст колонкой при нажатой левой кнопке (рис. 4.11). С выделенным куском текста можно проводить стандартные операции копирования, вырезания и вставки.

## Альтернативный метод поиска строк

Нажмите в редакторе кода комбинацию клавиш <Ctrl>+<I> и начинайте вводить любой текст. На экране появится значок с изображением бинокля (рис. 4.12). Теперь начните вводить символы с клавиатуры. При вводе начальных символов будут выделяться совпадающие символы первого вхождения строки от положения курсора и ниже.

```
public static string ReverseString(string str)
{
    // Проверка на существование строки.
    if (string.IsNullOrEmpty(str))
    {
        return str;
    }

    // Создадим объект StringBuilder с нужной длиной.
    StringBuilder revStr = new StringBuilder(str.Length);
    // Перебираем в цикле все символы
    // и присоединяем каждый символ к StringBuilder
    for (int count = str.Length - 1; count > -1; count--)
    {
        revStr.Append(str[count]);
    }
    // Возвращаем перевернутую строку
    return revStr.ToString();
}

private void butReverseCS_Click(object sender, EventArgs e)
{
    textBox1.Text = ReverseString(textBox1.Text);
```

**Рис. 4.11.** Выделение куска текста колонкой

```
public static string ReverseString(string str)
{
    // Проверка на существование строки.
    if (string.IsNullOrEmpty(str))
    {
        return str;
    }
    // Создадим объект StringBuilder с нужной длиной.
    StringBuilder revStr = new StringBuilder(str.Length);
    // Перебираем в цикле все символы
    // и присоединяем каждый символ к StringBuilder
    for (int count = str.Length - 1; count > -1; count--)
    {
        revStr.Append(str[count]);
    }
    // Возвращаем перевернутую строку
    return revStr.ToString();
}

private void butReverseCS_Click(object sender, EventArgs e)
{
    textBox1.Text = ReverseString(textBox1.Text);


}
```

**Рис. 4.12.** Поиск слова по первым символам

## Множественное копирование в буфер обмена

В Visual Studio 2008 используется собственный буфер обмена, позволяющий множественное копирование. Скопируйте нужный участок кода в буфер стандартным способом (через контекстное меню или сочетание клавиш <Ctrl>+<C>). После этого установите курсор в нужном месте редактора кода, нажмите и удерживайте клавиши <Ctrl>+<Shift> и затем нажимайте клавишу <V>. Вы увидите, что в указанном месте появляются фрагменты текста, которые вы ранее скопировали в буфер. Обратите внимание, что вставляемые части текста циклически чередуются.

## Как управлять фрагментами кода в Visual Studio 2008?

Если вы часто используете один и тот же фрагмент кода в своих проектах, то вам, несомненно, понравится возможность перетаскивания часто используемого кода на панель инструментов. В редакторе кода Visual Studio 2008 переключитесь в вид Code View, затем выберите ваш фрагмент кода и перетащите его при помощи мыши на вкладку панели инструментов. Теперь можно перетаскивать фрагмент из панели инструментов в нужное место кода (еще проще просто дважды щелкнуть по фрагменту). Так как я достаточно часто использую вызовы функций Windows API через `P/Invoke`, у меня там находится строка

```
using System.Runtime.InteropServices;
```

## Быстрое комментирование и раскомментирование фрагментов кода

Еще один альтернативный вариант для тех, кто привык работать с клавиатурой. Выделите фрагмент кода, который хотите закомментировать, и нажмите, удерживая клавишу <Ctrl>, клавиши <K> и <C>. Для снятия комментариев нужно выделить закомментированный кусок и нажать <Ctrl>+<K>, <U>. Члены общества любителей мышей могут просто выделить нужный текст и щелкнуть по специальному значку на панели инструментов.

## Отображение IntelliSense

Если вы хотите принудительно отобразить подсказку IntelliSense, нужно нажать <Ctrl>+<J>. Это может пригодиться в том случае, если по каким-то причинам подсказка не отображается.

## Прозрачная подсказка IntelliSense

Меня очень часто раздражало, что выпадающее окно подсказки IntelliSense закрывало часть кода, написанного строчками ниже. Приходилось прекращать печатать код, чтобы посмотреть, что же было написано в этих строчках, и потом снова возвращаться на редактируемую строку. Видимо, это раздражало не только меня, но и многих других программистов. В Visual Studio 2008 можно поступить следующим образом. Когда подсказка снова закроет нужную часть кода, то просто нажмите на клавишу <Ctrl>. Ух ты, подсказка стала прозрачной, и вы видите нужные данные! Отпустите клавишу, и подсказка снова примет свой стандартный вид. Снова нажмите на клавишу. Снова отпустите. Уверен, вам понравится. Теперь поводов для раздражения стало меньше, и можно наслаждаться написанием кода.

## Перемещение от открывающей скобки к закрывающей скобке

Иногда блок кода между фигурными скобками слишком велик и не умещается на экране. Чтобы быстро переместиться к закрывающей скобке класса, функции, цикла и т. п., установите курсор ввода перед скобкой и нажмите комбинацию клавиш <Ctrl>+<]>. Повторное нажатие этой комбинации вернет курсор к открывающей скобке. Этот способ работает также для многострочных комментариев (`/* */`), регионов (#region #endregion) и кавычек, обрамляющих строки.

## Сворачивание/разворачивание блока (региона, функции, цикла и т. п.)

В редакторе кода блоки исходного кода можно сворачивать, чтобы временно сократить место на мониторе и видеть общую структуру кода. Чтобы снова развернуть блок кода, не обязательно щелкать мышью по значкам +/−. Нажав комбинацию клавиш <Ctrl>+<M>, <M> (удерживая <Ctrl> нажать <M> два раза), вы сделаете то же самое с меньшими затратами. Комбинация <Ctrl>+ +<M>, <L> позволяет свернуть/развернуть все блоки.

## Анимация при автоматическом скрытии панелей

Если у вас не слишком мощный компьютер, то, возможно, вам стоит отключить эффект автоматически убирающихся окон, которые по умолчанию ис-

пользуются в Visual Studio. Этот эффект забирает много ресурсов у компьютера, что приводит к замедлению работы среды разработки. Настройка анимации окон находится в окне, открываемом командой меню **Tools** | **Options…**, на странице **General**. Здесь можно управлять скоростью анимации при автоматическом скрытии панелей. Скорость сворачивания зависит от положения ползунка **Speed**. Также вы можете совсем отключить анимацию, если уберете галочку **Animate Environment tools** (Анимировать средства среды).

## Вариант загрузки справочной системы

Если вы регулярно запускаете справочную систему, то замечали, что она ищет документацию не только на локальном компьютере, но и в Интернете. Вы можете отключить поиск в онлайн-источниках. В окне, открываемом командой меню **Tools** | **Options…**, на странице **Help** | **Online** нужно выбрать пункт **Try local only, not online** (Использовать только локальную, не онлайн). Такой выбор позволит быстрее отображать справку.

## Путь к файлу

Иногда требуется быстро получить полный путь к файлу, с которым вы работаете в IDE Visual Studio. Раньше приходилось открывать Проводник, искать нужный файл, смотреть его свойства и т. д. Теперь достаточно щелкнуть правой кнопкой на закладке с именем файла и выбрать пункт **Copy Full Path**.

## Быстрый переход к папке, содержащей исходные коды проекта

Впрочем, кроме получения полного пути к файлу часто требуется быстро открыть саму папку, в которой содержится нужный файл проекта. Если в Visual Studio 2008 у вас уже открыт проект, то вы можете открыть папку проекта прямо из среды разработки. Достаточно щелкнуть правой кнопкой мыши по вкладке любого файла, открытого в редакторе, и выбрать команду **Open Containing Folder** (рис. 4.13).

Эта команда позволяет мгновенно оказаться в нужной папке и произвести необходимые действия с файлом: поменять атрибуты файла, переименовать его и т. д.

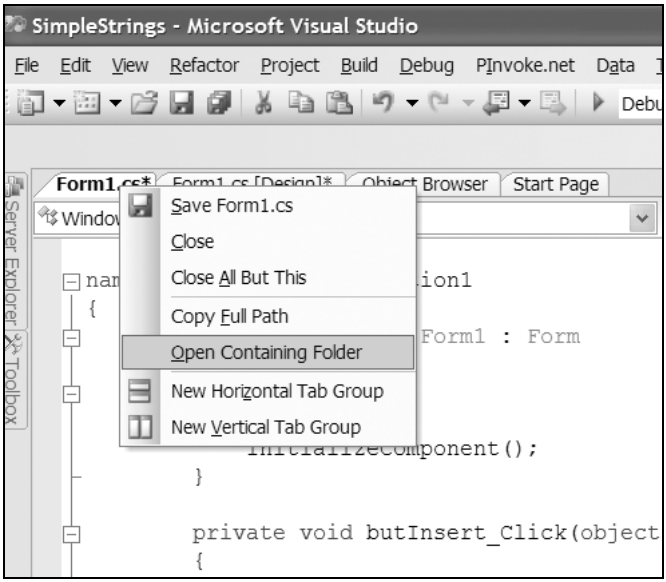



**Рис. 4.13.** Открываем папку, содержащую файл проекта

## Изменение шаблона заготовки метода в C#, генерируемого автоматически

Если вы в коде пишете имя еще не объявленного метода, то Smart Tag в Visual Studio предлагает создать заготовку метода. Выглядит это следующим образом. Предположим, в обработчике события `Click` для кнопки вы напечатали строку

```
PrintMyName();
```

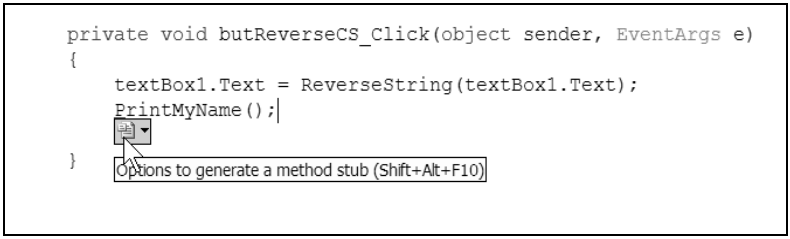



**Рис. 4.14.** Вставка заготовки для метода

Редактор кода выделит символом подчеркивания первый символ строчки кода, сигнализируя о том, что здесь имеется Smart Tag. Подведите указатель мыши к данному тегу (или используйте комбинацию клавиш <Alt>+<Shift>+<F10>) и активируйте строчку, предлагающую создать заготовку для метода (рис. 4.14). Среда сгенерирует несколько строк кода, которые вы можете отредактировать по собственному желанию. Если вас не устраивает стандартный вариант, то отредактируйте соответствующий файл Program Files\Microsoft Visual Studio 9\VC#\Snippets\1033\Refactoring\MethodStub.snippet.

## Вспомнить название пространства имен

Очень часто мы помним название класса, но не помним, к какому пространству имен относится данный класс. И здесь нам поможет интеллектуальная подсказка редактора кода Visual Studio. Просто введите в редакторе кода имя класса, и вы увидите, что редактор кода подчеркнул красной линией последнюю букву класса. Подведите указатель мыши к этому символу подчеркивания, чтобы развернуть специальное меню смарт-тега (рис. 4.15). Можно выбрать два варианта — либо импортировать пространство имен при помощи ключевого слова `using`, либо добавить нужное пространство имен перед классом. Особо хотелось бы остановиться на первом способе. Когда вы выбираете вариант импорта пространства имен, то строчка типа `using System.IO;` добавляется в начало вашего кода автоматически, и вам не придется прокручивать редактор кода в начало редактора кода. Это особенно удобно, если у вас очень большой проект, насчитывающий несколько тысяч строк кода.

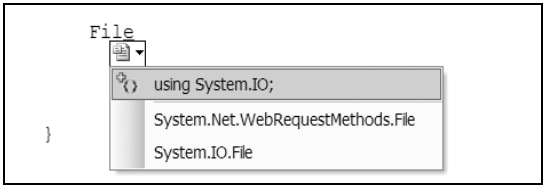



**Рис. 4.15.** Автоматическое добавление названия пространства имен

## Удобный способ вызвать Smart Tag

Вы увидели, что смарт-теги очень удобны для работы. Но, к сожалению, значок смарт-тега очень маленький, и мышкой по нему попасть достаточно за-

труднительно. Конечно, можно вызвать Smart Tag при помощи стандартной комбинации клавиш <Alt>+<Shift>+<F10>. Но, согласитесь, что удобной эту комбинацию назвать трудно. К счастью, среда разработки позволяет назначить собственное сочетание. Выберите меню **Tools | Customize…** и в открывшемся окне нажмите кнопку **Keyboard…**. Откроется окно настроек на странице настройки клавиатуры, и здесь в поле **Show command containing** нужно набрать ShowSmartTag. В результате этих действий в нижнем списке останется команда `View.ShowSmartTag`, которой можно сопоставить более удобное сочетание клавиш (рис. 4.16).

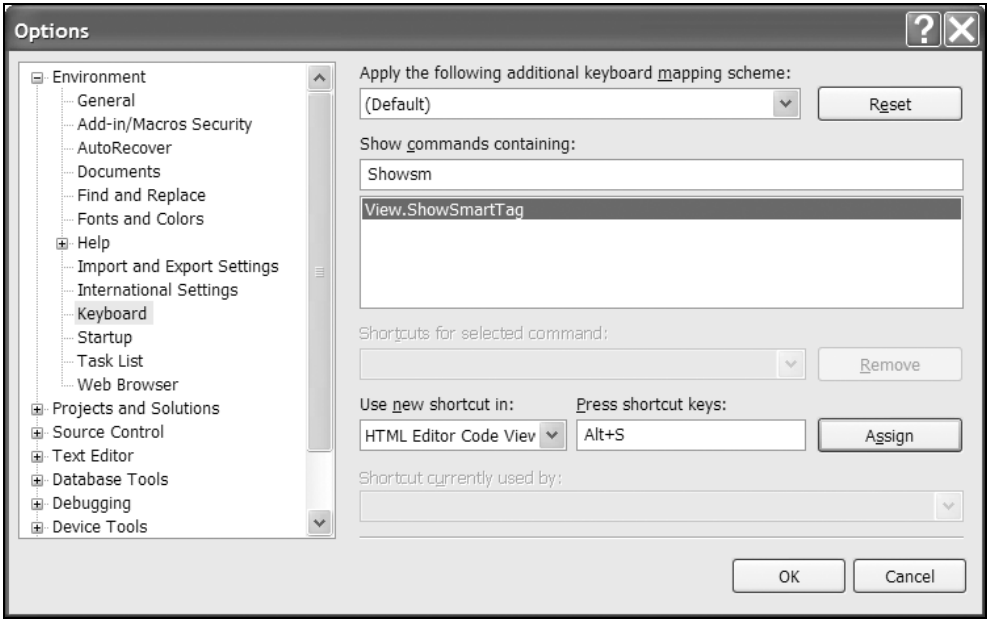



**Рис. 4.16.** Меняем комбинацию клавиш для Smart Tag

## Создание собственных шаблонов приложений

Вы можете добавить в список проектов Visual Studio свой собственный шаблон для приложения. Для этого вам нужно воспользоваться экспортом проекта как шаблона через команду **File | Export Template…**. Удобный мастер позволит вам за считанные секунды создать собственный шаблон, которым вы можете пользоваться в дальнейшем. По адресу **http://support.microsoft.com:80/kb/870715** вы

можете прочитать статью "How to create a custom code template in Visual Studio 2005 or in Visual Studio .NET" (на англ. языке), в которой вы найдете дополнительную информацию.

## Работа в полноэкранном режиме

Вы можете писать код, максимально расширив редактор кода. Чтобы работать в полноэкранном режиме, нужно нажать комбинацию клавиш <Alt>+ +<Shift>+<Enter>. Чтобы вернуться к прежнему состоянию, нужно еще раз нажать эту комбинацию клавиш.

## Быстрый поиск в списках

В окнах Visual Studio IDE очень часто используются различные списки. Причем количество элементов списков может быть огромным, и поиск нужного элемента становится утомительным занятием. Но, зная название нужного элемента, найти необходимую строчку достаточно просто. Приведем следующий пример. Предположим, вам нужно добавить в проект ссылку на пространство имен `Microsoft.VisualBasic`, чтобы задействовать некоторые полезные методы. Выберите в меню **Project | Add Reference...** и в открывшемся диалоговом окне начинайте вводить первые буквы нужной вам сборки — M. Список автоматически прокрутится до элемента, который начинается с этой буквы. Если вы обладаете большой скоростью печати, то можете продолжать вводить следующие буквы имени сборки. Увы, если вы не обладаете слепым десятипальцевым методом печати, то при наборе второй буквы I список прокрутится до элемента, который начинается на эту букву (если такой элемент списка имеется), так как программа будет думать, что вы начали новый поиск. Поэтому мой вам совет — учитесь печатать быстро.

# Поле *Find*

На панели инструментов чуть ниже основного меню имеется поле **Find**, предназначенное для быстрого поиска текста в коде программы. Чтобы попасть в него, можно нажать комбинацию клавиш <Ctrl>+</>. Но, на самом деле, функциональность окна гораздо шире. Например, введите в этом окне команду

```
>of
```

Далее сделайте пробел и начинайте вводить имя файла, входящего в проект. Откроется список с автозавершением, который поможет вам быстро ввести имя нужного файла (рис. 4.17). После того как вы выбрали нужный файл, просто нажмите на клавишу <Enter>, и код этого файла откроется в редакторе кода.

Это еще не все. Введите в этом же окне имя нового файла и нажмите комбинацию клавиш <Ctrl>+<N>. У вас будет создан новый файл с заданным именем, который нужно будет сохранить на диске.

А если вы введете в окне имя метода и нажмете клавишу <F9> вместо клавиши <Enter>, то напротив заданного метода будет установлена точка прерывания (breakpoint). Причем, если у вас в коде имеются перегруженные версии метода, то точки прерывания будут установлены у всех версий.

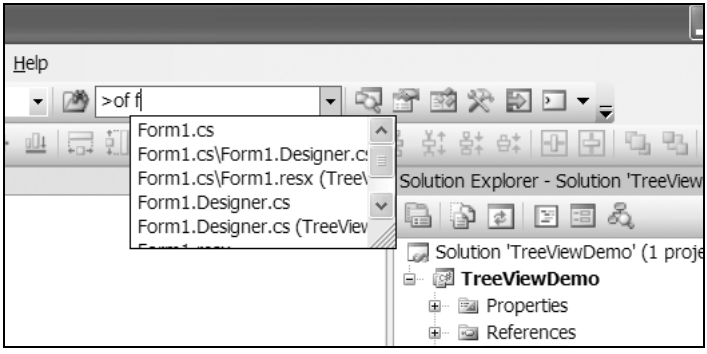



**Рис. 4.17.** Окно **Find**

# Окно *Command*

Но мы с вами не расстаемся с полем **Find** и продолжим опыты с ним. Данное поле тесно связано с окном **Command**, которое открывается с помощью комбинации <Ctrl>+<W>, <A>. И описываемые далее команды можно вводить в любом из них. Сначала поясним, что обозначает команда `>of` из предыдущего примера. Эта команда является сокращением-псевдонимом (alias) для команды **Open File**. Давайте попробуем ввести другие команды. Например, введите в окне **Find** или **Command** команду (регистр букв не имеет значения)

```
>AddProj
```

Когда вы нажмете на клавишу <Enter>, то у вас запустится диалоговое окно добавления нового проекта. В данном случае сокращение `AddProj` обозначает команду **Add New Project**. Теперь введите команду

```
>code
```

Если у вас был активным режим показа **Design Mode**, то после этой команды вы переключитесь в редактор кода. Данный псевдоним обозначает команду **View Code**. Если вы хотите узнать все имеющиеся псевдонимы, которыми можно воспользоваться в среде разработки, то наберите команду

```
>alias
```

Вы увидите список всех сокращений и обозначения этих команд (рис. 4.18).

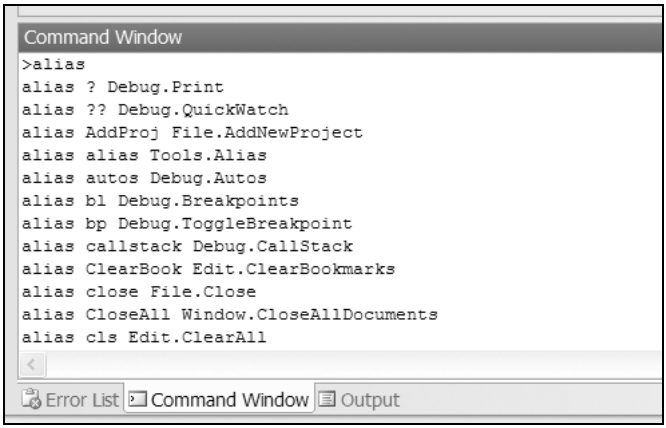



**Рис. 4.18.** Список команд-сокращений в окне **Command**

Более подробную информацию о псевдонимах, используемых в окнах **Find** и **Command**, можно почерпнуть из статьи MSDN "Predefined Visual Studio Command Aliases" по адресу **http://msdn2.microsoft.com/en-us/library/c3a0kd3x(VS.80).aspx**.

# Диалоговое окно *Find and Replace*

Вероятно, вам иногда приходилось пользоваться диалоговым окном **Find and Replace** для поиска того или иного слова. Но не многие знают, что в данном окне можно использовать регулярные выражения. Вызовите это диалоговое окно и отметьте галочкой строку **Use** с элементом списка **Regular**

**expressions**. У вас активируется кнопочка с треугольником **Expression Builder**, с помощью которой вы сможете сконструировать нужное регулярное выражение. Обратите внимание, что синтаксис регулярных выражений в этом окне несколько отличается от синтаксиса стандартных регулярных выражений, используемых в .NET Framework. Не совсем понятно, зачем понадобилось создавать собственную версию регулярных выражений для этого окна, но разработчикам Visual Studio виднее. Давайте создадим регулярное выражение, которое найдет все слова, начинающиеся на букву E:

```
<e[a-z]*
```

Проверим составленное выражение. Нажимаем на кнопку **Find Next** и видим, что выражение составлено правильно, и действительно находит слова, начинающиеся на букву E (рис. 4.19).

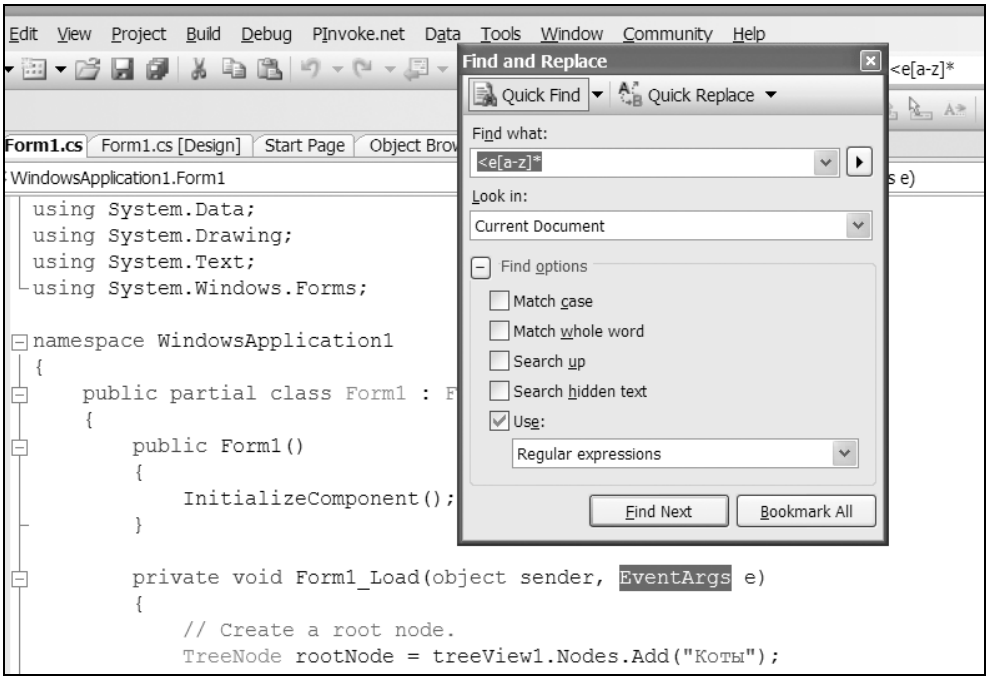



**Рис. 4.19.** Регулярное выражение в окне **Find and Replace**

Еще несколько советов по составлению регулярных выражений в диалоговом окне **Find and Replace** вы можете прочитать в блоге **http://blogs.msdn.com/vseditor/archive/2004/06/18/159515.aspx**.

# Еще о настройках

Рассмотрим еще несколько настроек, которыми можно управлять в окне, открываемом командой меню **Tools | Options…** на странице **Environment | General**.

## Сокрытие статусной строки

Вы можете скрыть статусную строку **StatusBar**, если уберете галочку с поля **Show Status Bar** в окне **Options**. Правда, трудно понять, зачем это может понадобиться.

## Число показываемых последних файлов

Число показываемых последних файлов можно настроить, указав желаемое значение в поле **Recent files**.

## Многодокументный интерфейс

По умолчанию используется модный сейчас интерфейс с использованием вкладок. Но вы можете переключиться на многодокументный интерфейс, если выберете переключатель **Multiple documents**.

## Управление панелями Auto Hide и Close

Почти все вспомогательные окна, которые окружают редактор кода или дизайнер форм, имеют две кнопки: булавку spin и крестик закрытия окна. За их поведение отвечают пункты

**Auto Hide button affects active tool window only**

**Close button affects active tool window only**

С помощью этих настроек можно управлять вспомогательными панелями. Например, можно настроить, будут ли автоматически прятаться все пристыкованные панели или это будет распространяться только на активную панель. Попробуйте самостоятельно поиграть этими значениями, чтобы понять, о чем идет речь.

## Меню *Window*

В меню **Window** есть команда **Auto Hide All**, которая заставит все окна автоматически закрываться, оставив на экране только основное окно редактора кода и дизайнера форм.

## Переключение между окнами

Подобно стандартному менеджеру переключения между окнами в Windows, вызываемому при нажатии клавиш <Alt>+<Tab>, в Visual Studio вы можете использовать комбинацию <Ctrl>+<Tab>. Данная комбинация также позволяет переключаться между активными окнами среды разработки, которые показываются в виде миниатюрных картинок.

## Помоги себе и команде Visual Studio, или пишем логи

Если запустить Visual Studio 2005 с параметром командной строки `Devenv.exe /log`, то в папке %USERPROFILE%\Application Data\Microsoft\Visual Studio\8.0\ будет создан файл ActivityLog.xml, содержащий лог работы Visual Studio. Данный лог нужно отослать разработчикам, которые смогут разобраться в найденном вами баге. Данный совет взят с блога Гайдара Магданурова **http://blogs.gotdotnet.ru/personal/gaidar/CommentView.aspx?guid=BFE909CE-8209-4E39-A2EF-9603B9AB2C2D#commentstart**.

## Графические файлы для проектов

Те, кто имел опыт работы с пакетом Visual Studio 6.0, помнят, что в его состав входил набор картинок, значков, курсоров, файлов анимации, которые можно было включать в свой проект, например, для создания стандартной панели инструментов с нужными пиктограммами. Такой же пакет имеется и в Visual Studio 2008, только он лежит в архивном файле VS2008ImageLibrary.zip по адресу C:\Program Files\Microsoft Visual Studio 9.0\Common7\VS2008ImageLibrary\1033. Вы можете распаковать данный архив и пользоваться этой коллекцией изображений. Обратите внимание, что в

каждой папке там имеется отдельный html-документ. Открыв его, вы увидите на веб-страничке сразу все картинки папки, что позволит вам быстро сориентироваться.

# Надстройки

В Visual Studio всегда имелась поддержка надстроек (add-in), которые позволяли расширить функциональность интегрированной среды разработки. Подобные дополнения можно написать самостоятельно или установить уже готовую продукцию от сторонних разработчиков.

# Надстройки сторонних разработчиков

Существует несколько полезных расширений, которые были разработаны сторонними разработчиками и получили известность и признание среди программистов. Я перечислю несколько популярных расширений, которые могут пригодиться вам в вашей практике.

## GhostDoc

Эта надстройка позволяет очень быстро создавать XML-комментарии при написании кода, которые потом можно обработать при помощи утилиты nDoc для создания файлов справки. GhostDoc позволяет автоматизировать процесс создания комментариев, экономя значительное время. Утилита просматривает имя вашего класса или метода, а также их параметров, и на основе этой информации пытается создать свой комментарий. Естественно, при именовании методов вам нужно придерживаться стандартных правил, принятых в программировании. Рассмотрим простой пример. Предположим, мы хотим написать обертку для метода `MessageBox` (листинг 4.8).

**Листинг 4.8. Создание обертки для метода `MessageBox`**

```
private void ShowMsg(string Message)
{
    MessageBox.Show(Message);
}
```

После установки GhostDoc можно щелкнуть правой кнопкой мыши на созданном методе и выбрать из контекстного меню пункт **Document this** (рис. 4.20). В результате перед методом появится комментарий, приведенный в листинге 4.9.

**Листинг 4.9. Комментарий, добавленный программой GhostDoc**

```
/// <summary>
/// Shows the MSG.
/// </summary>
/// <param name="message">The message.</param>
private void ShowMsg(string message)
{
    MessageBox.Show(message);
}
```

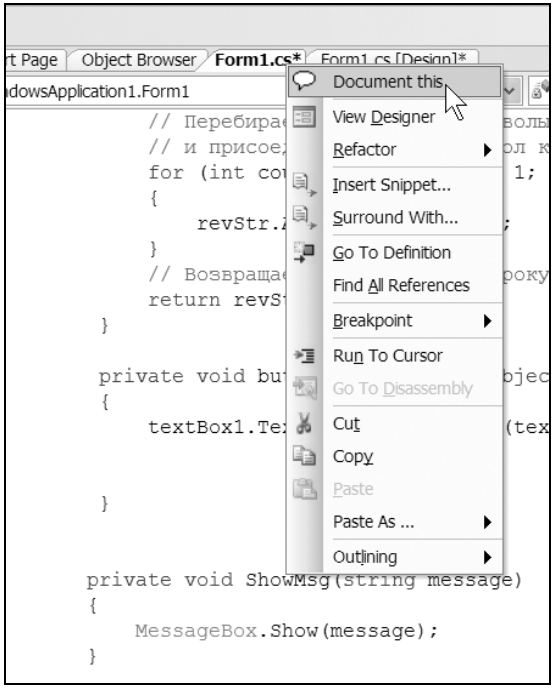



**Рис. 4.20.** Утилита GhostDoc в действии

Как видите, GhostDoc автоматически сгенерировала комментарий на основе имени метода `ShowMsg`, которое было разделено на словосочетание Shows the

MSG (помните о стандартах наименования?). Также были добавлены комментарии относительно параметра `message`. Естественно, после генерирования комментариев вы можете внести свои изменения для придания им большей ясности. Программа имеет окно настроек, где вы можете поменять правила создания комментариев или добавить собственные. Удобную бесплатную утилиту написал Роланд Вейгельт (Roland Weigelt), и скачать ее можно по адресу **http://www.roland-weigelt.de/ghostdoc**. GhostDoc работает с Visual Studio 2005 и Visual Studio 2008 (но не с Express-версиями). Автор программы любезно разрешил мне включить эту утилиту в компакт-диск, который прилагается к книге.

> ***Примечание***
>
> Утилиты GhostDoc для Visual Studio 2005 и 2008 находятся в папке GhostDoc на прилагаемом диске.

## SmartPaster

Название этого дополнения к IDE можно перевести как "изящная вставка". При написании кода для своего приложения часто приходится пользоваться вставкой уже готовых строк из другого источника. Причем обычная вставка из буфера обмена не всегда удобна. Предположим, вам нужно вставить в код десять строчек текста в виде комментария. Вот как это выглядит при стандартной ситуации: вы копируете кусок текста в буфер, например, из документа Word, вставляете этот текст в редактор кода, расставляете знаки комментариев на каждой строчке текста, подгоняете текст по ширине редактора и т. д. Дополнение SmartPaster избавит вас от этой головной боли. Достаточно щелкнуть правой кнопкой для вызова контекстного меню, в котором появится дополнительный пункт **Paste As**, выбрать эту команду и вставить текст в нужном формате. Скачайте архив, содержащий дополнение SmartPaster, переименуйте его в файл с раширением VSI и запустите его. Инсталлятор программы встроит SmartPaster в IDE пакета Visual Studio. Запустите любой проект, откройте редактор кода. Далее скопируйте из Блокнота кусочек текста в буфер обмена. Теперь посмотрим как работает программа. Выбираем из контекстного меню команду **Paste As | Comment**. В листинге 4.10 показано, что получится.

**Листинг 4.10. Результат вставки текста при помощи команды Paste As | Comment**

```
//Этот текст был набран в Word и скопирован в буфер обмена.
//Посмотрим, что у нас получится
//при использовании Add-in SmartPaster.
```

Как видите, к каждой строке текста был добавлен символ комментария. Вы можете настроить по желанию длину каждой строки и другие параметры. Попробуем другие варианты. При использовании команды **Paste As | String** получим код, приведенный в листинге 4.11.

**Листинг 4.11. Результат вставки текста при помощи команды Paste As | String**

```
string myString =
    @"Этот текст был набран в Word и скопирован в буфер обмена." +
    Environment.NewLine +
    @"Посмотрим, что у нас получится " + Environment.NewLine +
    @"при использовании Add-in SmartPaster.";
```

Как видите, утилита склеила все строки при помощи управляющего символа `@` и константы `Environment.NewLine`. Варианты с командами **StringBuilder** и **Region** изучите самостоятельно. Утилиту написал Алекс Пападимоулис (Alex Papadimoulis), который также выложил для изучения и исходный код программы. Найти надстройку SmartPaster можно по адресу **http://weblogs.asp.net/alex_papadimoulis/archive/2004/05/25/Smart-Paster-1.1-Add-In---StringBuilder-and-Better-C_2300_-Handling.aspx**.

## PInvoke.NET

PInvoke.NET — это одна из моих любимых надстроек в последнее время, учитывая мою любовь к функциям Windows API, которая тянется еще с изучения Visual Basic 5.0. Данная утилита (рис. 4.21) позволяет быстро получить объявление функции Windows API для дальнейшего использования. О функциях Windows API мы еще поговорим в следующих главах. А пока поговорим об этой утилите. Если вам нужно в своем проекте использовать вызов системной функции Windows API, то можно обойтись без поиска ее описания на бескрайних просторах Интернета. Установите PInvoke.NET, и у вас всегда под рукой будет удобный инструмент для вставки готового кода. Предположим, вам нужно найти объявление функции `ExitWindowsEx`. Выбираем пра-

вой кнопкой команду: **Insert PInvoke Signatures**, печатаем нужное название функции и получаем готовый результат (листинг 4.12).

**Листинг 4.12. Объявление функции API `ExitWindowsEx`**

```
[DllImport("user32.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)]
static extern bool ExitWindowsEx(ExitWindows uFlags,
ShutdownReason dwReason);
```

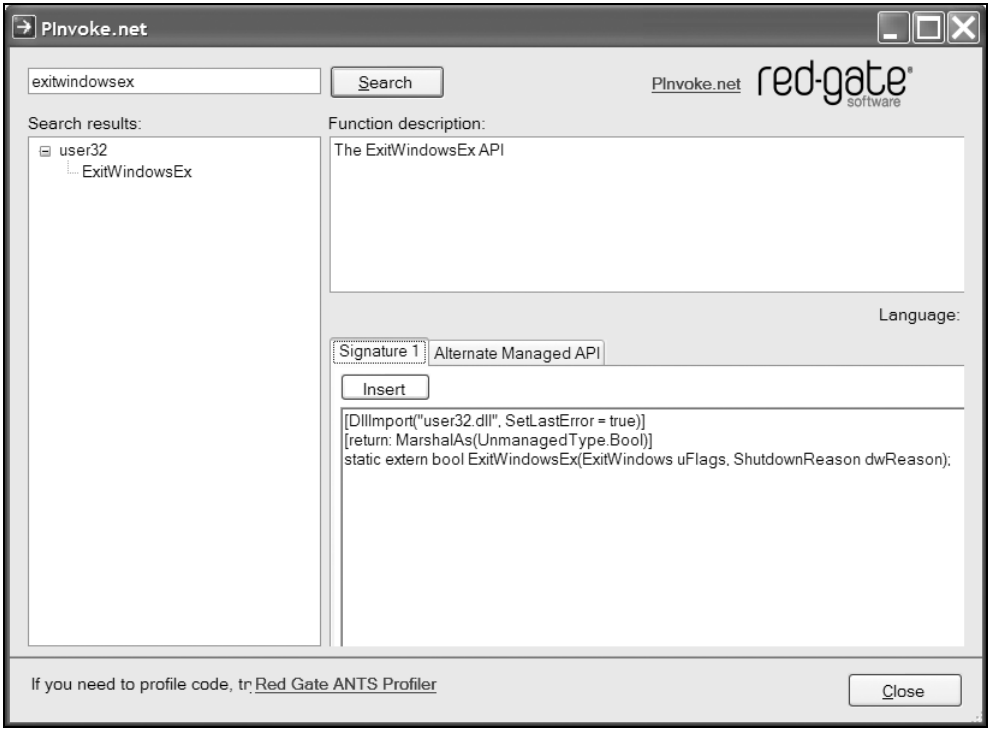



**Рис. 4.21.** Надстройка PInvoke.NET

Кстати, там имеется кнопка, позволяющая получить управляемый код (если он существует). Найти эту надстройку можно на сайте **http://www.pinvoke.net**. Только обязательно проверяйте полученное описание. Сайт **http://www.pinvoke.net** использует движок Википедии, и его содержимое определяют сами посетители сайта, которые могут добавлять и редактировать содержимое страниц с описанием функций. К сожалению, на

данный момент эта утилита не работает в Visual Studio 2008. Остается надеяться, что это временное явление и разработчики выпустят обновленную версию.

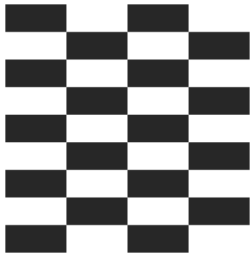
## Paste as Visual Basic

Если вам приходится писать программы как на C#, так и на Visual Basic, то советую обратить внимание на статью "Paste As Visual Basic: A Visual Studio Add-In That Converts C# Code To Visual Basic" из февральского выпуска журнала "MSDN Magazine" за 2006 год (электронная версия статьи находится по адресу **http://msdn.microsoft.com/msdnmag/issues/06/02/PasteAs/default.aspx**). \В этой статье говорится о том, как написать свою надстройу, а также по шагам описывается создание утилиты, которая на лету конвертирует код на C# в код на Visual Basic.

# Заключение

Мы рассмотрели только часть возможностей, заложенных в интегрированной среде разработки IDE Visual Studio. На самом деле среда разработки — это очень мощный инструмент, изучению которого можно посвятить много времени.

## ГЛАВА 5



# Экран и формы

Прежде чем мы перейдем к рассмотрению примеров для экрана и форм, чуть остановимся на процессе создания новых проектов в Visual Studio 2005/2008. Те, кто программировал в предыдущей версии Visual Studio .NET 2003, уже наверняка обратили внимание на изменения в структуре проекта. Из редактора кода формы были удалены генерируемые мастером кода строки кода по умолчанию. Если создать новый проект **Windows Application**, то в форме будет создан минимальный код, показанный в листинге 5.1.

**Листинг 5.1. Минимальный код для Windows Application**

```
using System.Text;
using System.Windows.Forms;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Естественно, разработчиков интересует вопрос, куда подевалась реализация `InitializeComponent`. Обратите внимание, что среда разработки создает помимо файла Form1.cs еще и файл Form1.Designer.cs, в котором и находится код формы, управляемый дизайнером Windows Forms. Чтобы разбить класс на два физических файла, разработчики Microsoft ввели поддержку частичных классов (partial classes), см. листинг 5.2.

**Листинг 5.2. Поддержка частичных классов**

```
    partial class Form1
    {
        // и так далее
```

Подобное разделение кода предотвращает случайное изменение кода, которое может привести к краху приложения, а также позволяет программисту сосредоточиться на написании собственного кода, решающего конкретные задачи.

Еще один вопрос, возникающий у программистов — куда подевалась точка входа в приложение, то есть метод `Main`? Поскольку статический метод `Main` не относится ни к какой форме, то разработчики Microsoft перенесли этот метод в отдельный файл Program.cs (листинг 5.3).

**Листинг 5.3. Код для файла Program.cs**

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

namespace WindowsApplication1
{
    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());
        }
    }
}
```

На этом краткое вступление можно считать законченным. Перейдем к советам, которые относятся к свойствам экрана и формы.

# Экран

## Как определить разрешение экрана

Для определения размеров экрана воспользуйтесь свойством `PrimaryScreen.Bounds` класса `Screen` из пространства имен `System.Windows.FormsScreen`, как показано в листинге 5.4.

**Листинг 5.4. Определение размеров экрана**

```
private void butScreenRes_Click(object sender, EventArgs e)
{
    // Определяем размеры экрана
    textBox1.Text = Screen.PrimaryScreen.Bounds.Width.ToString() +
        "x" + Screen.PrimaryScreen.Bounds.Height.ToString();
}
```

## Как определить рабочую область экрана без панели задач?

Иногда требуется узнать размеры не всего экрана, а только его рабочей части, в которую не входит панель задач, а также другие экранные панели (например, существовал пакет Microsoft Office, который использовал собственную панель). В этом случае вам поможет свойство `Screen.GetWorkingArea`, способное определить рабочую область экрана без панели задач. Как его использовать, показано в листинге 5.5.

**Листинг 5.5. Определение рабочей области экрана**

```
private void butWorkArea_Click(object sender, EventArgs e)
{
    // Определяем рабочую область экрана
    textBox1.Text = Screen.GetWorkingArea(this).Width.ToString() + "x" +
       Screen.GetWorkingArea(this).Height.ToString();
}
```

***Примечание***

Примеры находятся в папке AboutScreen на прилагаемом диске.

## Как изменить разрешение экрана программным путем

На данный момент .NET Framework не имеет встроенных методов для изменения разрешения экрана. Поэтому нам придется использовать две функции Windows API: `EnumDisplaySettings` и `ChangeDisplaySetting`. Описание этих функций есть в справочнике по функциям API на прилагаемом к книге компакт-диске, поэтому ограничимся только кодом программы, приведенным в листинге 5.6.

**Листинг 5.6. Программное изменение разрешения экрана**

```
using System.Runtime.InteropServices;

[DllImport("user32.dll")]
public static extern int EnumDisplaySettings(
  string deviceName, int modeNum, ref DEVMODE devMode);

[DllImport("user32.dll")]
public static extern int ChangeDisplaySettings(
      ref DEVMODE devMode, int flags);

[StructLayout(LayoutKind.Sequential)]
public struct DEVMODE
{
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 32)]
    public string dmDeviceName;
    public short dmSpecVersion;
    public short dmDriverVersion;
    public short dmSize;
    public short dmDriverExtra;
    public int dmFields;

    public short dmOrientation;
    public short dmPaperSize;
    public short dmPaperLength;
    public short dmPaperWidth;
```

```
    public short dmScale;
    public short dmCopies;
    public short dmDefaultSource;
    public short dmPrintQuality;
    public short dmColor;
    public short dmDuplex;
    public short dmYResolution;
    public short dmTTOption;
    public short dmCollate;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 32)]
    public string dmFormName;
    public short dmLogPixels;
    public short dmBitsPerPel;
    public int dmPelsWidth;
    public int dmPelsHeight;

    public int dmDisplayFlags;
    public int dmDisplayFrequency;

    public int dmICMMethod;
    public int dmICMIntent;
    public int dmMediaType;
    public int dmDitherType;
    public int dmReserved1;
    public int dmReserved2;

    public int dmPanningWidth;
    public int dmPanningHeight;
};

public const int ENUM_CURRENT_SETTINGS = -1;
public const int CDS_UPDATEREGISTRY = 0x01;
public const int CDS_TEST = 0x02;
public const int DISP_CHANGE_SUCCESSFUL = 0;
public const int DISP_CHANGE_RESTART = 1;
public const int DISP_CHANGE_FAILED = -1;

private void button1_Click(object sender, EventArgs e)
{
    Screen screen = Screen.PrimaryScreen;
```

```
// Разрешение 640 на 480
int iWidth = 640;
int iHeight = 480;

DEVMODE dm = new DEVMODE();
dm.dmDeviceName = new String (new char[32]);
dm.dmFormName = new String (new char[32]);
dm.dmSize = (short)Marshal.SizeOf (dm);

if (0 != EnumDisplaySettings(null, ENUM_CURRENT_SETTINGS, ref dm))
{
    dm.dmPelsWidth = iWidth;
    dm.dmPelsHeight = iHeight;

    int iRet = ChangeDisplaySettings(ref dm, CDS_TEST);

    if (iRet == DISP_CHANGE_FAILED)
    {
        MessageBox.Show("Не получается поменять разрешение");
    }
    else
    {
        iRet = ChangeDisplaySettings(ref dm, CDS_UPDATEREGISTRY);

        switch (iRet)
        {
            case DISP_CHANGE_SUCCESSFUL:
            {
                break;
                // разрешение успешно поменяли
            }
            case DISP_CHANGE_RESTART:
            {
                MessageBox.Show("Нужно перезагрузить компьютер");
                break;
                // для старых версий windows 9x
            }
            default:
            {
                MessageBox.Show("Ошибка при изменении разрешения");
```

```
                    break;
                    // попытка не удалась
                }
            }
        }
    }
}
```

***Примечание***

Пример находится в папке ChangeResolution на прилагаемом диске.

# Формы

## Как вывести форму в центре экрана?

У формы есть свойство `StartPosition`, которому вы можете сопоставить необходимое значение. Например, чтобы вывести форму точно по центру экрана, используйте значение `CenterScreen`. Но возможна ситуация, когда вы установили значение данного свойства равным `Manual`, но, тем не менее, хотите, чтобы форма появилась в центре экрана. В таком случае вам придется самостоятельно вычислять необходимые координаты для формы. Добавьте код, приведенный в листинге 5.7, для события `Load` формы.

**Листинг 5.7. Вывод формы в центре экрана**

```
private void Form1_Load(object sender, EventArgs e)
{
    // Выводим форму в центре экрана
    this.Location = new Point((Screen.PrimaryScreen.Bounds.Width -
                                                    this.Width) / 2,
        (Screen.PrimaryScreen.Bounds.Height - this.Height) / 2);
}
```

## Как задать позицию формы на экране?

Пусть, например, перед вами стоит задача установить форму в правом нижнем углу над панелью задач. Для этой цели очень хорошо подходит связка свойств `SetBounds` и `GetWorkingArea`. Необходимый для этого код показан в листинге 5.8.

**Листинг 5.8. Задаем позицию формы на экране**

```
// Выводим окно в нижнем правом углу экрана
this.SetBounds(Screen.GetWorkingArea(this).Width - this.Width,
    Screen.GetWorkingArea(this).Height - this.Height,
    this.Width, this.Height);
```

## Как программно свернуть или развернуть форму?

У формы есть свойство `WindowState`, которое поможет вам решить эту задачу. Например, чтобы свернуть форму, добавьте код из листинга 5.9.

**Листинг 5.9. Программное сворачивание формы**

```
// Свернем форму программно
WindowState = FormWindowState.Minimized;
```

## Поддержка тем рабочего стола Windows

Обратите внимание, что в .NET Framework 2.0 при создании проектов с использованием Windows Forms поддержка визуальных тем рабочего стола Windows XP включена по умолчанию. Если вернуться чуть-чуть назад, то мы увидим, что в .NET Framework 1.0 для использования визуальных тем приходилось писать специальный файл манифеста, а в .NET Framework 1.1 этот процесс немножко упростился благодаря новому методу `Application.EnableVisualStyles`. В Visual Studio 2005 программисты Microsoft пошли еще дальше. Метод `EnableVisualStyles` вставляется автоматически в каждый новый проект. Убедиться в этом вы можете самостоятельно, открыв код файла Program.cs, приведенный в листинге 5.10.

**Листинг 5.10. Поддержка тем рабочего стола Windows (код из файла Program.cs)**

```
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());
}
```

## Как узнать, используются ли темы Windows XP?

Чтобы узнать, используются ли на компьютере визуальные темы XP или пользователь предпочитает работать с классическим видом, можно воспользоваться помощью функции Windows API `IsThemeActive`. Как это сделать, показано в листинге 5.11.

**Листинг 5.11. Использование тем Windows XP**

```
[DllImport("uxtheme.dll")]
static extern bool IsThemeActive();

// Выводим сообщение в заголовке формы
this.Text = "Тема XP: " + IsThemeActive().ToString();
```

## Как отобразить форму без передачи ей фокуса?

В .NET Framework 2.0 у форм появилось новое свойство `ShowWithoutActivation`. Название свойства говорит само за себя. Если при выводе на экран окно не активно, то свойство возвращает `True`, в противном случае `False`. По умолчанию окно получает фокус (то есть значение свойства равно `False`). Иногда требуется вывести окно без передачи ему фокуса, чтобы не мешать пользователю продолжать работу с основным окном. Вспомните стандартный графический редактор Paint из состава Windows. Если вы выбираете инструмент **Надпись**, то появляется окно, позволяющее настроить шрифт. При этом новое окно не отбирает фокус. Учтите, что свойство доступно только для чтения. Вы можете изменить значение свойства, только переопределив свойство `ShowWithoutActivation` в своей форме. Для иллюстрации того, как это сделать, создадим две формы. На первой (и основной) форме добавим кнопку, щелчок по которой будет вызывать вторую форму (листинг 5.12).

**Листинг 5.12. Показываем вторую форму**

```
private void button2_Click(object sender, EventArgs e)
{
    // Показываем вторую форму
    Form2 secondform = new Form2();
    secondform.Show();
}
```

Если мы сейчас запустим пример и щелкнем по кнопке, то вторая форма откроется и станет активной. Переопределим свойство `ShowWithoutActivation` во второй форме, чтобы решить проблему.

**Листинг 5.13. Переопределение свойства ShowWithoutActivation**

```
protected override bool ShowWithoutActivation
{
    get
    {
        return true;
    }
}
```

Запустите проект и убедитесь, что вторая форма теперь не получает фокус.

> ***Примечание***
>
> Если вы вынуждены писать программы с использованием .NET Framework 1.1, то вам придется использовать функцию Windows API `ShowWindow` с параметром `SW_SHOWNA`.

## Как не отображать форму при запуске программы?

Вы хотите запустить приложение, не показывая саму форму? Первое, что приходит в голову — присвоить свойству формы `Visible` значение `False`. Проблема только в том, что такого свойства нет в редакторе свойств. Более правильное решение — установить форму невидимой во время события `Load`. Но при таком решении, все равно, форма на какое-то время появится на экра-

не и затем исчезнет. Все дело в файле Program.cs. Откройте его и увидите три строчки:

```
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new Form1());
```

Код в первой строке делает доступным использование визуальных тем XP для элементов управления. Вторая строчка отключает рендеринг текста с использованием старой версии GDI+ из .NET Framework 1.1 (подробнее это описано в блоге **http://blogs.msdn.com/jfoscoding/archive/2005/10/13/480632.aspx**). И, наконец, третья строчка — ключ решения нашей задачи. Мы видим, что метод `Run` ссылается на главную форму, которая становится формой по умолчанию. Но в нашем случае мы не хотим, чтобы она была формой по умолчанию. Мы не можем настроить это поведение через окно свойств, но можем изменить код:

```
Form1 mainForm = new Form1();
Application.Run();
```

Запустив проект, вы увидите, что никакая форма не появлялась на экране, но, тем не менее, наша программа выполняется.

> ***Примечание***
>
> Пример со скрытой формой находится в папке NoForm на прилагаемом диске.

## Как сделать так, чтобы форма отбрасывала тень?

Если вы хотите, чтобы ваша форма отбрасывала тень, то переопределите свойство `CreateParams`, добавив к свойству `ClassStyle` новое значение `CS_DROPSHADOW`, как показано в листинге 5.14.

**Листинг 5.14. Тень для формы**

```
private const int CS_DROPSHADOW = 0x00020000;

// Переопределяем свойства CreateParams
protected override CreateParams CreateParams
{
    get
```

```
    {
        CreateParams cp = base.CreateParams;
        cp.ClassStyle |= CS_DROPSHADOW;
        return cp;
    }
}
```

Запустив проект еще раз и вызвав на экран второе окно, вы увидите, что теперь оно отбрасывает тень. Имейте в виду, что данный код предназначен для операционных систем Windows XP и выше, так как старые версии Windows не поддерживают подобную функциональность на уровне системы. В этом случае вам придется самим позаботиться об отрисовке подобного эффекта.

## Как вывести запрос при закрытии формы?

Иногда встречаются ситуации, когда необходимо удостовериться, что пользователь действительно хочет закрыть программу, а не по ошибке щелкнул на кнопке закрытия программы. В этом случае нужно написать код для события `FormClosing`, показанный в листинге 5.15.

**Листинг 5.15. Вывод сообщения при закрытии формы**

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (MessageBox.Show("Вы уверены, что хотите выйти из программы?",
                        "Внимание!", MessageBoxButtons.YesNo)
        == DialogResult.No)
    {
        e.Cancel = true;
    }
}
```

Типичная ситуация, когда можно использовать это событие, — наличие текстового поля, содержащего измененный текст, который необходимо сохранить перед закрытием программы. В этом случае вы предоставляете пользователю шанс, чтобы он сначала сохранил изменения, а уже потом снова закрыл программу. Обратите внимание, что событие `FormClosing` пришло на смену событию `Closing` из .NET Framework 1.1, которое считается устаревшим. Если у вас остались старые проекты, то не забудьте сделать соответствующие изменения в них. В последнее время многие программы используют

событие FormClosing для сворачивания в область уведомлений в виде значка, а реальное закрытие формы происходит при выборе команды контекстного меню приложения. Подобное поведение, в частности, характерно для программ обмена мгновенными сообщениями ICQ, Qip, Mail@Agent. О том, как реализовать подобную функциональность, будет рассказано в *главе 6*, когда речь пойдет об элементе управления NotifyIcon.

## Выбираем варианты закрытия формы

Можно усложнить пример и сделать поведение при закрытии формы более избирательным. В .NET Framework 2.0 появилось новое свойство FormClosingEventArgs.CloseReason, которое позволяет выбрать способ закрытия окна при помощи перечисления System.Windows.Forms.CloseReason. Это перечисление имеет следующие поля:

- ApplicationExitCall — был вызван метод Exit класса Application;
- FormOwnerClosing — была закрыта родительская форма;
- MdiFormClosing — была закрыта родительская форма MDI (окно с многодокументным интерфейсом);
- None — не был определен способ закрытия формы;
- TaskManagerClosing — приложение было закрыто через Диспетчер задач (Windows Task Manager);
- UserClosing — окно закрывается пользователем через элементы пользовательского интерфейса: через элементы управления, кнопку **Закрыть (×)** на заголовке формы, команду **Закрыть** в системном меню или нажатием клавиш <Alt>+<F4>;
- WindowsShutDown — приложение закрывается операционной системой при выключении системы.

Ниже мы рассмотрим пример, который позволяет заблокировать кнопку **X** на заголовке формы. А здесь мы пойдем другим путем. Кнопка закрытия окна останется доступной, но пользователь все равно не сможет закрыть окно этим способом. Текст программы приведен в листинге 5.16.

**Листинг 5.16. Попытка закрытия формы**

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (e.CloseReason == CloseReason.UserClosing)
```

```
    {
        MessageBox.Show("Не положено!");
        e.Cancel = true;
    }

}

private void button1_Click(object sender, EventArgs e)
{
    // Все равно не закроется!
    this.Close();
}

private void button2_Click(object sender, EventArgs e)
{
    // А так закроется
    Application.Exit();
}
```

Итак, мы выбрали вариант `CloseReason.UserClosing`, когда окно формы закрывается пользователем. В этом случае мы игнорируем его действия и выдаем соответствующее сообщение. Для сравнения я поместил на форме две кнопки с разными способами завершения работы. При методе `this.Close` также используется значение `UserClosing` перечисления, препятствующее закрытию окна, а при вызове метода `Application.Exit` программа спокойно закрывается.

## Сокрытие значка формы на панели задач и при нажатии комбинации клавиш <Alt>+<Tab>

Если вы хотите, чтобы значок вашей программы не отображался на панели задач, то просто установите свойство формы `ShowInTaskBar` в значение `false`. Это можно сделать в дизайнере форм или программным способом (листинг 5.17), поместив соответствующий код в конструктор формы.

**Листинг 5.17. Скрытие значка формы на панели задач**

```
public MyForm()
{
    ShowInTaskbar = false;
}
```

Несмотря на то, что на панели задач значок формы уже не выводится, его все равно можно увидеть при вызове окна переключения задач (task switch) с помощью комбинации клавиш <Alt>+<Tab>. Чтобы избежать появления значка и в этом окне, присвойте свойству `FormBorderStyle` значение `SizableToolWindow`:

```
FormBorderStyle = FormBorderStyle.SizableToolWindow;
```

## Как отобразить форму на весь экран?

Иногда требуется, чтобы форма занимала весь экран, закрывая в том числе и панель задач. Типичные примеры для такой задачи — создание хранителей экрана или игр в полноэкранном режиме. Решить эту проблему очень просто. Присвойте свойству формы `FormBorderStyle` значение `FormBorderStyle.None`, а свойству `WindowState` значение `FormWindowState.Maximized`. Установить требуемые свойства вы можете в дизайнере форм или программным способом, показанным в листинге 5.18.

**Листинг 5.18. Отобразить форму на весь экран**

```
public Form1()
{
  FormBorderStyle = FormBorderStyle.None;
  WindowState = FormWindowState.Maximized;
}
```

## Как установить ограничение на минимальный и максимальный размер окна?

Если вы хотите, чтобы пользователь мог менять размеры окна в определенных пределах, то используйте в этих целях свойства формы `Form.MinimumSize` и `Form.MaximumSize`. Это можно сделать в дизайнере форм или программным способом, используя код листинга 5.19.

**Листинг 5.19. Установка ограничения на размер формы**

```
MaximumSize = new Size(400, 400);
MinimumSize = new Size(250, 250);
```

## Как отловить момент сворачивания или разворачивания формы?

Чтобы отловить момент сворачивания, разворачивания или восстановления формы, нужно перехватывать сообщение Windows `WM_SYSCOMMAND` и анализировать параметр `wParam`, который содержит необходимую информацию, а именно флаги `SC_MINIMIZE`, `SC_MAXIMIZE` и `SC_RESTORE`. Кроме того, необходимо следить за двойными щелчками мыши по заголовку формы, при которых форма также меняет свое состояние. В этом случае мы следим за сообщением `WM_NCLBUTTONDBLCLK`. Обратите внимание, что уведомления посылаются до изменения состояния формы, и мы имеем возможность отменить действие. Соответствующий код приведен в листинге 5.20.

**Листинг 5.20. Отлавливаем моменты изменения состояния формы**

```
const int WM_SYSCOMMAND = 0x0112;
const int SC_MINIMIZE = 0xF020;
const int SC_MAXIMIZE = 0xF030;
const int SC_RESTORE = 0xF120;
const int WM_NCLBUTTONDBLCLK = 0x00A3;
const int HTCAPTION = 2;
public string strInfo;

protected override void WndProc(ref Message msg)
{
    bool isStateChanging = false;

    switch (msg.Msg)
    {
        case WM_SYSCOMMAND:
            switch (msg.WParam.ToInt32())
            {
                case SC_MINIMIZE:
                    isStateChanging = true;
                    strInfo = "Сворачиваемся";
                    break;
                case SC_MAXIMIZE:
                    strInfo = "Разворачиваемся";
                    isStateChanging = true;
                    break;
```

```
                case SC_RESTORE:
                    strInfo = "Восстанавливаемся";
                    isStateChanging = true;
                    break;
            }
            break;
        case WM_NCLBUTTONDBLCLK:
            if (msg.WParam.ToInt32() == HTCAPTION)
            {
                isStateChanging = true;
                if (WindowState == FormWindowState.Maximized)
                    strInfo = "Восстанавливаемся";
                else
                    strInfo = "Разворачиваемся";
            }
            break;
    }
    if (isStateChanging)
        MessageBox.Show(strInfo);
    base.WndProc(ref msg);
}
```

## Как запретить пользователю перемещать форму по экрану?

Самый простой способ, который приходит в голову для решения этой задачи — установить стиль границы `FormBorderStyle` в значение `None`. В этом случае у формы не будет заголовка, и пользователь не сможет перемещать форму по экрану. Но внешний вид формы в этом случае оставляет желать лучшего. Есть еще более хитрый вариант. Установите свойства `ControlBox`, `MinimizeBox` и `MaximizeBox` формы в значение `false`. Затем присвойте свойству `Text` пустую строку. У формы исчезнет заголовок, но при этом появится граница вокруг формы. Внешне подобное окно выглядит более аккуратным. Ну а что делать, если необходимо, чтобы была неподвижной стандартная форма с обычным заголовком, и пользователь не мог бы перемещать ее по экрану? В этом случае нужно отлавливать некоторые системные сообщения в процедуре `WndProc` (листинг 5.21).

**Листинг 5.21. Запрет на перемещение формы пользователем**

```
protected override void WndProc(ref Message m)
{
    const int WM_NCLBUTTONDOWN = 161;
    const int WM_SYSCOMMAND = 274;
    const int HTCAPTION = 2;
    const int SC_MOVE = 61456;

    if((m.Msg == WM_SYSCOMMAND) && (m.WParam.ToInt32() == SC_MOVE))
    {
        return;
    }

    if((m.Msg == WM_NCLBUTTONDOWN) && (m.WParam.ToInt32() == HTCAPTION))
    {
        return;
    }

    base.WndProc (ref m);
}
```

***Примечание***

Примеры находятся в папке FormDemo на прилагаемом диске.

## Как перемещать форму, не имеющую заголовка?

В предыдущем примере мы запрещали пользователю перемещать форму. А теперь нам надо решить обратную задачу. Пусть у нас имеется форма без заголовка, которую нужно перемещать с помощью мыши. Реализуется эта задача следующим образом. Поместим на форму метку `Label`, которая возьмет на себя функцию заголовка по перемещению формы. Далее в редакторе кода объявим глобальную переменную типа `Boolean`, которая отслеживает, осуществляется ли в настоящий момент перетаскивание. Когда пользователь нажмет на кнопку мыши внутри `Label`, то эта переменная получит значение `true`, свидетельствующее о попытке пользователя перетащить форму. В этот момент нужно получить координаты мыши и обработать эту информацию

(код для события `MouseDown`). Далее, когда пользователь начинает перемещать мышь над `Label`, форма движется в том же направлении с таким расчетом, чтобы позиция мыши относительно `Label` оставалась неизменной (код для события `MouseMove`). Наконец, когда пользователь отпускает кнопку мыши, режим перетаскивания отключается (код для события `MouseUp`). В листинге 5.22 показано, как это выглядит в коде.

**Листинг 5.22. Перемещение формы без заголовка**

```
// Объявляем переменную, которая включает режим перетаскивания
private bool bDragStatus;

// Хранит координаты смещения при щелчке мышью
private Point clickPoint;

private void label1_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        // Включаем режим перетаскивания
        // и сохраняем координаты мыши
        bDragStatus = true;
        clickPoint = new Point(e.X, e.Y);
    }
    else
    {
        bDragStatus = false;
    }
}

private void label1_MouseMove(object sender, MouseEventArgs e)
{
    if (bDragStatus)
    {
        Point pointMoveTo;
        // Получим текущие координаты мыши в экранных координатах
        pointMoveTo = this.PointToScreen(new Point(e.X, e.Y));

        // Изменяем позицию на величину clickPoint
        pointMoveTo.Offset(-clickPoint.X, -clickPoint.Y);
```

```
        // Перемещаем форму
        this.Location = pointMoveTo;
    }
}

private void label1_MouseUp(object sender, MouseEventArgs e)
{
    // Отключаем режим перетаскивания
    bDragStatus = false;
}

private void button1_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Кстати, не забывайте в подобных формах предусмотреть возможность выхода из программы. Так как в созданном окне нет ни кнопки закрытия программы, ни команд системного меню, то пользователь просто не сможет закрыть вашу программу, и вспомнит вас недобрым словом.

> ***Примечание***
>
> Пример находится в папке DragForm на прилагаемом диске.

## Еще два способа буксировки формы, не имеющей заголовка

Существует еще один способ перемещения формы, не имеющей заголовка. На этот раз (листинг 5.23) мы воспользуемся вызовами функций Windows API.

**Листинг 5.23. Перетаскивание формы при помощи функций Windows API**

```
[DllImportAttribute("user32.dll")]
public static extern int SendMessage(IntPtr hWnd,
                int Msg, int wParam, int lParam);

[DllImportAttribute("user32.dll")]
public static extern bool ReleaseCapture();
```

```
public const int WM_NCLBUTTONDOWN = 0xA1;
public const int HT_CAPTION = 0x2;

private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        ReleaseCapture();
        SendMessage(this.Handle, WM_NCLBUTTONDOWN, HT_CAPTION, 0);
    }
}
```

***ПРИМЕЧАНИЕ***

Пример находится в папке DragFormAPI на прилагаемом диске.

Этот пример был простой калькой со старого примера, который я использовал в Visual Basic 6.0. Но оказалось, можно было обойтись без вызовов неуправляемого кода, если воспользоваться методом `WndProc`. Этот пример описан на сайте Microsoft по адресу **http://support.microsoft.com/kb/320687/** в статье "How to move a form by dragging the client area of the form" (на англ. языке).

**Листинг 5.24. Еще один способ перетаскивания формы без неуправляемого кода**

```
private const int WM_NCHITTEST = 0x84;
private const int HTCLIENT = 0x1;
private const int HTCAPTION = 0x2;

protected override void WndProc(ref Message m)
{
    switch (m.Msg)
    {
        case WM_NCHITTEST:
            base.WndProc(ref m);
            if ((int)m.Result == HTCLIENT)
                m.Result = (IntPtr)HTCAPTION;
            return;
            break;
    }
    base.WndProc(ref m);
}
```

***Примечание***

Пример с перетаскиванием формы находится в папке DragWndProc на прилагаемом диске.

## Как добиться эффекта полупрозрачности у формы

Одно из самых замечательных новшеств, которые появились в платформе .NET Framework, это поддержка прозрачности. Чтобы достичь подобного эффекта в старых версиях C++ или Visual Basic 6.0, требовалось писать уйму кода с использованием функций Windows API. Теперь же установка уровня прозрачности является встроенным свойством формы. Достаточно установить нужную степень прозрачности через свойство `Opacity` (всего лишь одна строчка кода!). Для большей наглядности приведем несколько полезных советов по использованию прозрачности в приложениях. Для начала посмотрим, как пользователь может управлять степенью прозрачности. Создадим новую форму и разместим на ней элемент управления `NumericUpDown`. Также добавим кнопку и таймер для реализации очень красивого эффекта. Код программы будет выглядеть следующим образом.

**Листинг 5.25. Устанавливаем уровень прозрачности для формы**

```
// управляем степенью прозрачности с помощью
// набора счетчиков NumericUpDown
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    this.Opacity = (double)numericUpDown1.Value / 100;
}
```

Щелкая по набору счетчиков `NumericUpDown`, пользователь может регулировать степень прозрачности формы. Но можно изменять прозрачность через таймер. Предположим, мы хотим, чтобы форма при закрытии исчезала не сразу, а постепенно, создавая иллюзию растворения в воздухе. Для данного эффекта нам и понадобится таймер, который мы поместили в проект ранее. При нажатии на кнопку таймер активируется и меняется степень прозрачности формы. Как только величина прозрачности достигнет значения 0, мы закрываем форму. Код приведен в листинге 5.26.

**Листинг 5.26. Эффект затухания при выходе из программы**

```
private void butClose_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;
}

private void timer1_Tick(object sender, EventArgs e)
{
    this.Opacity -= 0.1;
    if (this.Opacity <= 0)
        this.Close();
}
```

## Перемещение формы за заголовок

Рассмотрим еще один пример с прозрачностью формы (листинг 5.27). Например, когда мы будем перетаскивать форму за ее заголовок с помощью мыши, то форма временно станет полупрозрачной.

**Листинг 5.27. Использование прозрачности при перемещении формы**

```
protected override void WndProc(ref Message m)
{
    const int WM_NCLBUTTONDOWN = 0x00A1;
    const int WM_NCMOUSEMOVE = 0x00A0;


    if (m.Msg == WM_NCLBUTTONDOWN)
    {
        this.Opacity = 0.5;
    }

    if (m.Msg == WM_NCMOUSEMOVE)
    {
         this.Opacity = 1.0;
     }

     base.WndProc(ref m);
 }
```

Запустите проект и потаскайте форму по экрану. Вы увидите, что при перемещении форма становится полупрозрачной.

## Неактивная форма

Вот вам еще один пример с прозрачностью формы. Когда на экране находятся несколько открытых форм, то это может вызвать ощущение некоторого дискомфорта. Я предлагаю вам следующий вариант (листинг 5.28). Можно отловить сообщение о том, что окно программы стало неактивным, и сделать его полупрозрачным. И наоборот, когда окно вновь станет активным, вернуть ему первоначальное состояние.

**Листинг 5.28. Прозрачность у неактивной формы**

```
protected override void WndProc(ref Message m)
{
    const int WM_ACTIVATEAPP = 0x001C;

    if (m.Msg == WM_ACTIVATEAPP)
    {
        if (m.WParam.ToInt32() != 0)
        {
            this.Opacity = 1.0;
        }
        else
        {
            this.Opacity = 0.5;
        }
    }

    base.WndProc(ref m);
}
```

***Примечание***

Примеры находятся в папке TransparentDemo на прилагаемом диске.

## Как создать формы без границ и заголовка?

Установите соответствующие значения в свойствах `Text` и `ControlBox`, как показано в листинге 5.29.

**Листинг 5.29. Создание формы без границ и заголовка**

```
form1.Text = string.Empty;
form1.ControlBox = false;
```

## Как убрать кнопку *X* из заголовка формы?

Еще один популярный вопрос, задаваемый на форумах — как убрать кнопку **X** в правом верхнем углу заголовка формы? Как ни странно, эта кнопка тесно связана с системным меню, точнее с последним ее пунктом **Закрыть**. Если мы уберем этот пункт из меню, то кнопка **X** на форме примет заблокированный вид. Для реализации этой задачи (листинг 5.30) нам понадобится функция Windows API `GetSystemMenu`, а также функции `GetMenuItemCount`, `RemoveMenu` и `DrawMenuBar`....

**Листинг 5.30. Объявление функций Windows API для системного меню**

```
// Флаги для системного меню
public enum MenuFlags
{
    MF_BYCOMMAND = 0x00000000,
    MF_BYPOSITION = 0x00000400,
}

[DllImport("user32.dll")]
private static extern IntPtr GetSystemMenu(IntPtr hwnd,
int bRevert);

[DllImport("user32.dll")]
private static extern int GetMenuItemCount(IntPtr hMenu);

[DllImport("user32.dll")]
private static extern int RemoveMenu(IntPtr hMenu,
int Position, MenuFlags uFlags);
```

```
[DllImport("user32.dll")]
private static extern bool DrawMenuBar(IntPtr hWnd);
```

Как правило, команда **Закрыть** является последней командой системного меню, которой предшествует разделитель. Таким образом, нам нужно просто подсчитать число всех команд меню и удалить две последние команды (листинг 5.31). Только не забудьте предусмотреть выход из программы, ведь если кнопка **X** и команда **Закрыть** будут заблокированы, то пользователь не сможет выйти из программы.

**Листинг 5.31. Убираем кнопку X из заголовка формы**

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

    // Получаем описатель системного меню
    IntPtr hSysMenu = GetSystemMenu(this.Handle, 0);
    int menucount = GetMenuItemCount(hSysMenu);

    // Убираем последний пункт меню - команду Закрыть
    RemoveMenu(hSysMenu, menucount - 1, MenuFlags.MF_BYPOSITION);

    // Убираем предпоследний пункт меню - разделитель
    RemoveMenu(hSysMenu, menucount - 2, MenuFlags.MF_BYPOSITION);

    // Перерисовываем меню
    DrawMenuBar(hSysMenu);
}

private void butExit_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

***Примечание***

Пример находится в папке RemoveX на прилагаемом диске.

# Убрать кнопку X при помощи управляемого кода

Долгое время я думал, что убрать кнопку **X** из заголовка формы можно только при помощи неуправляемого кода с привлечением системных функций Windows API. Но однажды, при посещении сайта **http://www.codeproject.com**, я обнаружил пример, в котором показывалась возможность убрать эту кнопку, используя только управляемый код. Все оказалось очень просто (листинг 5.32), нужно было просто внимательно почитать документацию MSDN.

**Листинг 5.32. Убираем кнопку X при помощи управляемого кода**

```
private const int CS_NOCLOSE = 0x200;

protected override CreateParams CreateParams
{
    get
    {
        CreateParams cp = base.CreateParams;
        cp.ClassStyle = cp.ClassStyle | CS_NOCLOSE;
        return cp;
    }
}

private void butExit_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Как видите, нам нужно просто переопределить свойство `CreateParams`, добавив к свойству `ClassStyle` новое значение `CS_NOCLOSE`.

> ***Примечание***
>
> Пример находится в папке NoClose на прилагаемом диске.

# Создать окно произвольной формы

Существует несколько способов для реализации этой идеи. Рассмотрим очень быстрый и простой способ. У формы имеется свойство `TransparencyKey`, кото-

рое определяет цвет прозрачности. Зальем всю поверхность формы каким-нибудь цветом, определим его как прозрачный цвет и далее рисуем любую фигуру другими цветами (листинг 5.33). Вот как будет выглядеть форма в виде ромба (рис. 5.1).

**Листинг 5.33. Создание формы в виде ромба**

```
public Form1()
{
    InitializeComponent();
    // Задаем цвет прозрачности
    this.TransparencyKey = Color.Gold;
}

private void Form1_Paint(object sender, PaintEventArgs e)
{
    e.Graphics.Clear(Color.Gold);

    e.Graphics.FillPolygon(Brushes.Aquamarine, new PointF[] {
        new Point(this.ClientRectangle.Width / 2, 0),
        new Point(this.ClientRectangle.Width,
                  this.ClientRectangle.Height / 2),
        new Point(this.ClientRectangle.Width / 2,
                  this.ClientRectangle.Height),
        new Point(0, this.ClientRectangle.Height / 2)});

    Font f = new Font(this.Font.Name, 20);
    e.Graphics.DrawString("C#.Народные советы",
        f, Brushes.Black, 20,
        (this.ClientRectangle.Height / 2) - (f.Height / 2));
}
```

Для большей наглядности мы оставили заголовок и границы формы. Поэтому наша форма выглядит в виде ромба, вписанного в прямоугольник. Но ничего не мешает сделать вам действительно форму в виде ромба. Попробуйте сами изменить пример.

***Примечание***

Пример находится в папке DiamondForm на прилагаемом диске.

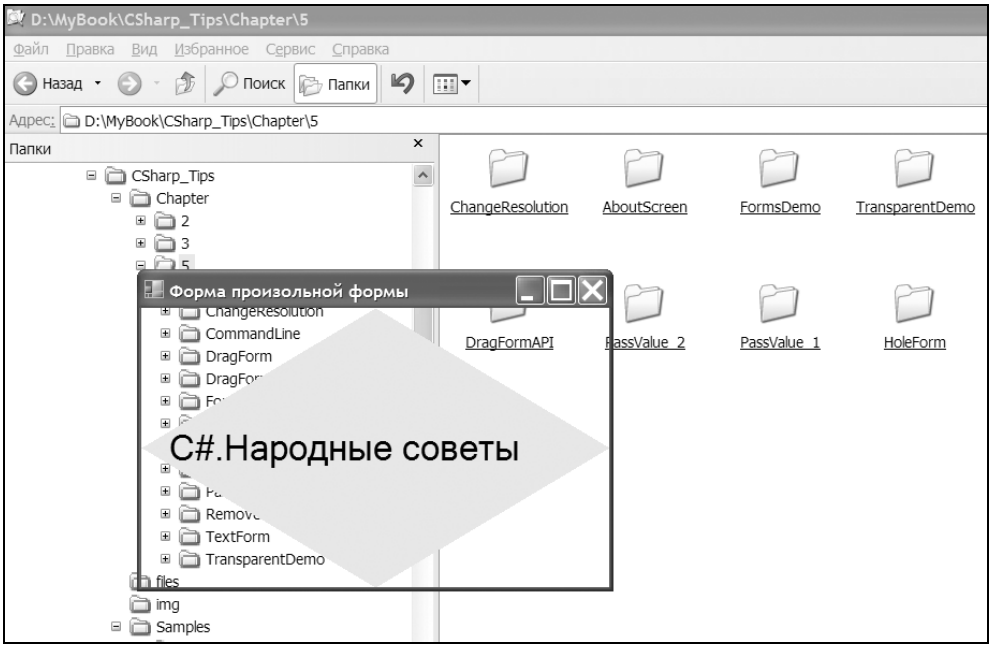



**Рис. 5.1.** Форма в виде ромба

## Создание дырявой формы

Очень необычно смотрятся не только окна произвольной формы, но и дырявые окна. Реализовать такое окно можно всего за несколько секунд, написав пару строчек кода. Размещаем на форме элемент `PictureBox`, который послужит нам в качестве отверстия в форме. Вот и все приготовления, и осталось добавить код из листинга 5.34.

**Листинг 5.34. Создание дырявой формы**

```
public Form()
{
    InitializeComponent();
    this.pictureBox1.BackColor = System.Drawing.Color.FromArgb(
        ((System.Byte)(255)),
        ((System.Byte)(128)),
        ((System.Byte)(128)));
```

```
    this.TransparencyKey = System.Drawing.Color.FromArgb(
        ((System.Byte)(255)),
        ((System.Byte)(128)),
        ((System.Byte)(128)));
}
```

Запустив проект, вы увидите стандартную прямоугольную форму с отверстием посередине (рис. 5.2).

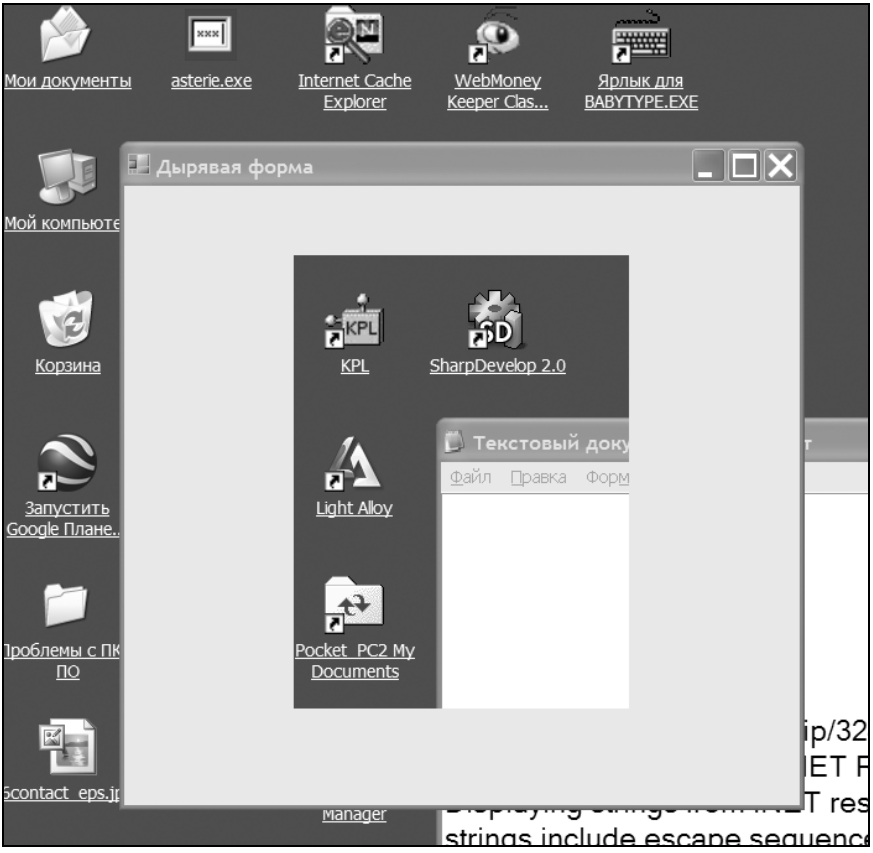



**Рис. 5.2.** Дырявая форма на Рабочем столе

### *Примечание*

Проект находится в папке HoleForm на прилагаемом диске.

## Как создать форму в виде текста?

В предыдущем примере мы использовали прозрачность, чтобы создать окно нестандартной формы. Есть еще один вариант создания произвольной фигуры для формы при помощи свойства `Region`. Например, можно создать форму в виде текста. Как это сделать, показано в листинге 5.35.

**Листинг 5.35. Форма в виде текста**

```
using System.Drawing.Drawing2D;

private void Form1_Paint(object sender, PaintEventArgs e)
{
    this.BackColor = Color.Cyan;
    GraphicsPath gp = new GraphicsPath();
    string myText = "C#.Народные советы";
    gp.AddString(myText, new FontFamily("Tahoma"),
        (int)FontStyle.Bold, 70, new Point(35, 5),
        StringFormat.GenericDefault);

    this.Region = new Region(gp);
}

private void Form1_DoubleClick(object sender, EventArgs e)
{
    this.Close();
}
```

В этом примере сначала форме присваивается цвет `Cyan` и создается экземпляр класса `GraphicsPath`. В качестве текста, форму которого примет наше окно, я выбрал название книги. Далее в методе `AddString` устанавливаются необходимые свойства для шрифта — название, стиль, координаты вывода и формат. Теперь достаточно присвоить свойству `Region` созданную траекторию, и все лишнее будет отброшено, а останется только текст. Обратите внимание, что часть заголовка формы попала в созданный регион. Таким образом, вы можете перетащить форму на другое место, вызвать системное меню и т. д. Если вы создадите форму без заголовка или регион не захватит часть заголовка, то вам необходимо предусмотреть возможность выхода из программы. В нашем примере форма закрывается при двойном нажатии мыши.

***Примечание***

Пример находится в папке TextForm на прилагаемом диске.

## Смена темы Windows XP

Ранее мы уже рассматривали пример, в котором определяли наличие визуальных стилей XP. Но существует еще сообщение `WM_THEMECHANGED`, которое посылается всем окнам при смене, активации и деактивации визуальных стилей Windows XP. Таким образом, мы можем отслеживать это сообщение (листинг 5.36), чтобы поймать момент смены тем на компьютере. Для того чтобы переключиться в классический вид, нужно открыть апплет Панели управления **Экран**, выбрать вкладку **Оформление** и в списке **Окна и кнопки** выбрать **Классический стиль**.

**Листинг 5.36. Отслеживание смены визуальных тем Windows XP**

```
const int WM_THEMECHANGED = 0x031A;

protected override void WndProc(ref Message m)
{
    // выводим сообщение
    // при смене визуальных стилей
    if (m.Msg == WM_THEMECHANGED)
        MessageBox.Show ("Стиль изменился!");
    base.WndProc (ref m);
}
```

В .NET Framework 2.0 появился новый класс `VisualStyleInformation`, в котором содержатся свойства, связанные с визуальными стилями XP. Среди прочих есть свойство `IsEnabledByUser`, с помощью которого можно определить, используется ли визуальный стиль XP текущим пользователем. Заодно (листинг 5.37) посмотрим и на остальные свойства класса (рис. 5.3).

**Листинг 5.37. Просмотр свойств класса `VisualStyleInformation`**

```
private void Form1_Load(object sender, EventArgs e)
{
    listBox1.Items.Add(VisualStyleInformation.Author);
    listBox1.Items.Add(VisualStyleInformation.ColorScheme);
```

```
    listBox1.Items.Add(VisualStyleInformation.Company);
    listBox1.Items.Add(VisualStyleInformation.Copyright);
    listBox1.Items.Add(VisualStyleInformation.Description);
    listBox1.Items.Add(VisualStyleInformation.DisplayName);
    listBox1.Items.Add(VisualStyleInformation.IsEnabledByUser);
}
```

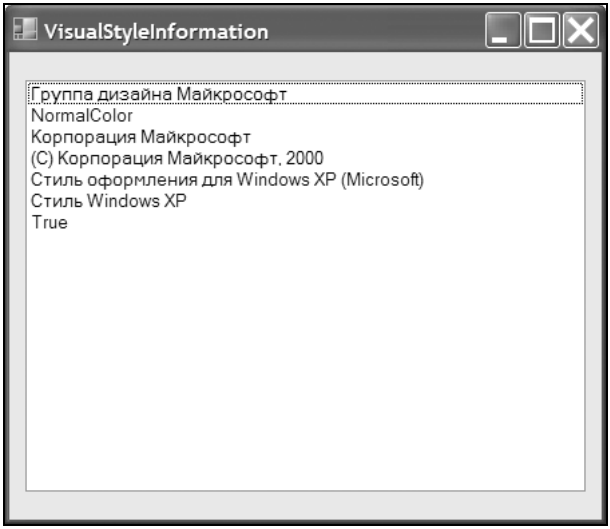



**Рис. 5.3.** Просмотр свойств класса `VisualStyleInformation`

***Примечание***

Пример находится в папке VisualStyleInfo на прилагаемом диске.

## Как форме получать уведомления о нажатии кнопок, когда фокус ввода находится в каком-либо элементе управления формы?

Это элементарная задача, она решается установкой свойства формы `Form.KeyPreview` в значение `true`. Для чего это нужно? Например, можно создать так называемое пасхальное яйцо. Предположим, мы зашиваем в программу какое-нибудь кодовое слово. Если пользователь нажмет клавиши в нужной последовательности, то на экран выпрыгивает какое-нибудь секрет-

ное окно с фотографией. Для простоты примера ограничимся стандартным `MessageBox` (листинг 5.38).

**Листинг 5.38. Использование свойства `KeyPreview`**

```
public static string TestString = "";

private void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    const string EASTER_EGG = "ТЕЩА";

    TestString = TestString + e.KeyChar.ToString().ToUpper();
    this.Text = TestString;
    if (EASTER_EGG.Substring(0, TestString.Length) != TestString)
    {
        TestString = "";
    }

    else
    {
        if (EASTER_EGG == TestString)
        {
            MessageBox.Show("Возьми с полки пирожок");
            TestString = "";
        }
    }
}
```

***Примечание***

Пример находится в папке EasterEgg на прилагаемом диске.

## Как получить список всех открытых форм, принадлежащих приложению?

Вам необходимо получить доступ ко всем открытым формам, которые принадлежат вашему приложению? В .NET Framework 2.0 появилось свойство `OpenForms`, которое относится к классу `Application`. Вам нужно пройти в цикле через объект `FormCollection` для получения свойства `OpenForms`. Почитайте документацию.

## Сохранение настроек формы

В .NET Framework 2.0 появились удобные механизмы сохранения настроек для формы. Эти настройки применяются в том случае, когда необходимо запомнить некоторые свойства при новом запуске программы. Настройками можно управлять как в режиме дизайна, так и программно. Конфигурирование программных настроек осуществляется через редактор свойств. Выберите любой элемент управления и найдите в редакторе свойств свойство `ApplicationSettings`. Раскройте это свойство и щелкните на `PropertyBinding`. У вас откроется окно настроек выбранного элемента. При конфигурировании свойств вам необходимо установить имя, значение по умолчанию и видимость для свойства. Имя позволяет получить доступ к свойству программным путем, значение по умолчанию позволяет использовать заранее подготовленные значения в том случае, если приложение не может получить необходимые настройки при запуске программы. Видимость определяет режим настроек. Программные настройки (Application scope) хранятся в конфигурационном файле, который обычно находится в одной папке с приложением, и доступны только для чтения. Вы можете менять настройки, редактируя конфигурационный файл без необходимости перекомпиляции программы. Настройки User scope доступны для чтения и записи и хранятся в профиле пользователя. Рекомендую самостоятельно изучить этот вопрос для использования в своих проектах.

## Создание и использование параметров командной строки

Для получения параметров командной строки существует метод `Environment.GetCommandLineArgs`. Давайте сначала создадим полусекретное приложение (листинг 5.39), которое будет запускаться только при наличии параметра командной строки.

**Листинг 5.39. Использование командной строки**

```
private void Form1_Load(object sender, EventArgs e)
{
    if (Environment.GetCommandLineArgs().Length > 1)
    {   // если есть аргумент командной строки
        if (Environment.GetCommandLineArgs()[1] == "csharp")
        { // если первый аргумент оканчивается на csharp
            this.Text = "Зарегистрированный пользователь";
```

```
        }
        else
        {
            MessageBox.Show("У вас нет прав для запуска программы");
            this.Close();
        }
    }
    else
    {
        // Запретить запуск программы
        MessageBox.Show("У вас нет прав для запуска программы");
        this.Close();
    }
}
```

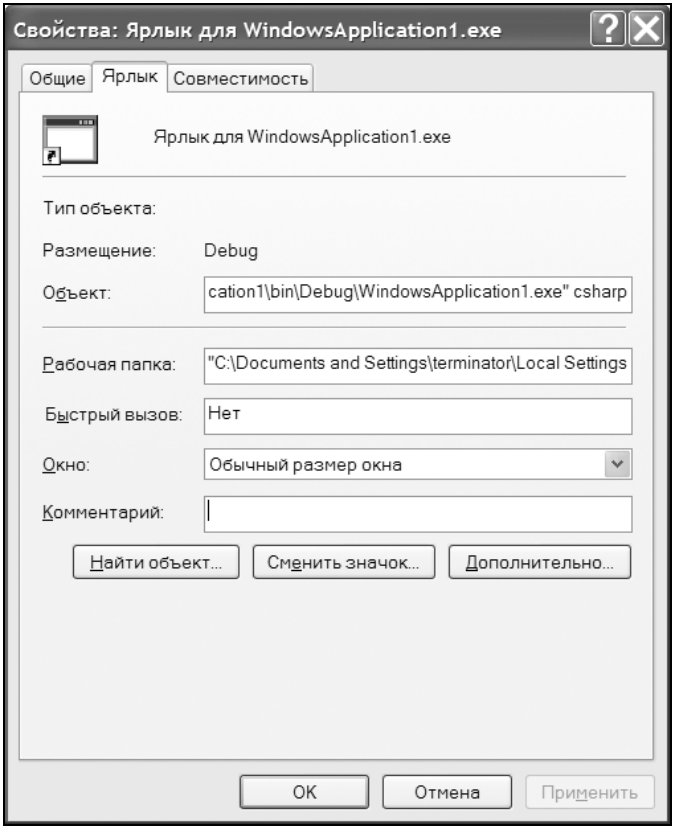



**Рис. 5.4.** Добавляем параметр к командной строке

Создайте исполняемый файл проекта и попробуйте запустить его. У вас появится окно сообщения, что у вас нет достаточных прав, после чего программа закроется. Теперь создайте ярлык к созданному исполняемому файлу и в свойствах ярлыка добавьте к файлу программы параметр `csharp` (рис. 5.4). Еще раз попробуйте запустить программу через ярлык. Программа без проблем запустится.

Следующая наша задача — узнать параметр командной строки. Добавляем строки кода, приведенные в листинге 5.40.

**Листинг 5.40. Получение параметров командной строки**

```
private void button1_Click(object sender, EventArgs e)
{
    String[] arguments = Environment.GetCommandLineArgs();
    textBox1.Text = arguments[1];
}
```

Теперь после щелчка по кнопке в текстовом поле отобразится параметр командной строки.

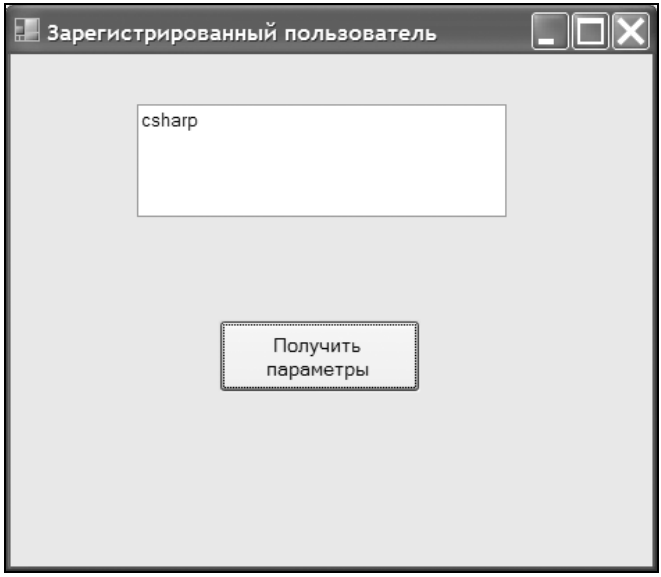



**Рис. 5.5.** Получение параметров командной строки

***ПРИМЕЧАНИЕ***

Пример находится в папке CommandLine на прилагаемом диске.

## Установить фоновый цвет в родительской MDI-форме

Если ваше приложение использует многодокументный интерфейс (MDI), то вам будет интересно узнать, как можно изменить фоновый цвет в родительской MDI-форме. Сначала создайте обычный проект и присвойте свойству `BackColor` формы нужный вам цвет, например, оранжевый (`Orange`). Далее присвойте свойству `IsMDIContainer` значение `True`. Обратите внимание, что при этом фон формы станет соответствовать цвету, определенному в системе для окна программы. Далее установите свойство `WindowState` в значение `Maximized`, чтобы окно программы открывалось в раскрытом виде. Также добавьте в проект новую форму `Form2`, которая будет являться дочерним окном программы. Теперь, чтобы изменить фоновый цвет родительского MDI-окна программным способом, напишем следующий код (листинг 5.41) для события `Load` родительской формы.

**Листинг 5.41. Меняем фоновый цвет в MDI-форме**

```
private void Form1_Load(object sender, EventArgs e)
{
    MdiClient ctlMDI;

    // Проходим через все элементы формы
    // в поисках типа MdiClient.
    foreach (Control ctl in this.Controls)
    {
        // Пытаемся привести элемент к типу MdiClient
        ctlMDI = (MdiClient)ctl;

        // Устанавливаем цвет BackColor элемента MdiClient
        ctlMDI.BackColor = this.BackColor;
    }

    // Выводим дочернюю форму в окне родительской MDI-формы
    Form2 frm = new Form2();
    frm.MdiParent = this;
    frm.Show();
}
```

***Примечание***

Пример работы с MDI-формой находится в папке MDIBackColor на прилагаемом диске.

## Запрет на запуск второй копии приложения

Многие приложения могут запускаться только в единственном экземпляре. Такова логика приложения. Например, программа Outlook Express запускается только в одном экземпляре. Чтобы реализовать эту функциональность в своем проекте, нам нужно воспользоваться классом `Mutex` в методе `Main`. Как это сделать, показано в листинге 5.42.

**Листинг 5.42. Запуск одного экземпляра приложения**

```
using System.Threading;

static void Main()
{
    bool oneonly;
    // Указываем имя своей программы. В нашем случае My prog
    Mutex m = new Mutex(true, "My prog", out oneonly);

    if (oneonly)
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        //Application.Run(new Form1());

        Form1 mainForm = new Form1();
        Application.Run();
    }
    else
    {
        MessageBox.Show("Это приложение уже запущено.");
    }
}
```

***Примечание***

Пример запрета на запуск второго экземпляра находится в папке SingleInstance на прилагаемом диске.

# Как передавать значения между формами Windows Forms

Например, требуется передать текст из текстового поля одной формы в текстовое поле другой формы. Есть два способа передачи значений некоторых данных между формами. Рассмотрим оба варианта.

## Способ первый

Добавим на форму текстовое поле и кнопку. Мы хотим, чтобы при нажатии на кнопку текст из текстового поля был передан на вторую форму. Первое важное замечание. Когда вы поместили на форму текстовое поле `textBox1`, то оно по умолчанию имеет модификатор доступа `private`. Вам необходимо изменить этот модификатор на `public`, иначе вторая форма не сможет получить данные из первой формы. Для этого откройте код файла Form1.Designer.cs и измените одну строчку.

```
//private System.Windows.Forms.TextBox textBox1;    // было по умолчанию
public System.Windows.Forms.TextBox textBox1;       // измененная строка
```

Далее напишем код (листинг 5.43) для кнопки.

**Листинг 5.43. Показ второй формы**

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 form2 = new Form2();
    // form1 является экземпляром класса Form1 в Form2.cs,
    // определенный как public
    form2.form1 = this;
    // Показываем форму Form2
    form2.ShowDialog();
}
```

Переходим на форму `Form2`. Откроем редактор кода и напишем сразу после конструктора объявление переменной `form1` и далее напишем код, который будет выполняться при загрузке второй формы.

**Листинг 5.44. Передача текста из первой формы во вторую**

```
public Form1 form1;


private void Form2_Load(object sender, EventArgs e)
{
    this.txtGetText.Text = ((Form1)this.form1).textBox1.Text;
}
```

Теперь вы можете запустить проект и проверить, как все работает. Введите любой текст в текстовом поле первой формы и щелкните на кнопке **Передать**. Введенный вами текст появится во второй форме.

> ***ПРИМЕЧАНИЕ***
>
> Рассмотренный проект находится в папке PassValue_1 на компакт-диске.

## Второй способ

Второй способ заключается в использовании конструктора для передачи данных. Создадим новый проект, аналогичный предыдущему. Откройте окно редактора кода для второй формы. Конструктор по умолчанию для `Form2` выглядит, как показано в листинге 5.45.

**Листинг 5.45. Конструктор формы `Form2`**

```
public Form2()
{
    InitializeComponent();
}
```

Мы добавим еще один свой новый конструктор. Перегруженная версия конструктора будет принимать строковый параметр.

**Листинг 5.46. Перегруженная версия конструктора**

```
public Form2(string strParam)
{
    InitializeComponent();
    this.txtGetText.Text = strParam;
}
```

Вернемся к первой форме и напишем код для кнопки, после нажатия которой данные с текстового поля первой формы будут переданы во вторую форму.

**Листинг 5.47. Передача данных другой форме**

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 frm2 = new Form2(this.textBox1.Text);
    frm2.ShowDialog();
}
```

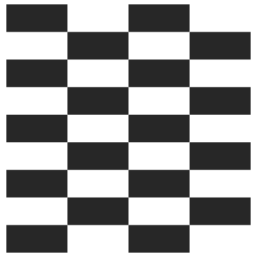
Запустите проект, щелкните на кнопке и убедитесь, что данные были успешно переданы во вторую форму.

> ***ПРИМЕЧАНИЕ***
>
> Этот пример находится в папке PassValue_2 прилагаемого компакт-диска.

# Заключение

В этой главе мы рассмотрели базовые приемы для работы с формами. Теперь пора двигаться дальше и научиться работать с объектами на форме.

ГЛАВА 6



# Элементы управления

Пространство имен `System.Windows.Forms` является одним из наиболее ключевых компонентов в .NET Framework. Оно содержит огромное количество классов, перечислений, методов и т. п. И его мощь продолжает увеличиваться. Достаточно сказать, что по сравнению с .NET Framework 1.1 в .NET Framework 2.0 количество открытых типов и членов пространства имен `System.Windows.Forms` увеличилось, по словам разработчиков, более чем на 60% и 120% соответственно. Элементы управления являются частью этого пространства имен. Число новых элементов управления растет, а прежние элементы обзаводятся новой функциональностью. Рассмотрим часть этих возможностей.

## Общие советы

### Как добавить элемент управления на форму во время выполнения программы?

Чтобы добавить любой элемент управления динамически во время исполнения программы, придерживайтесь следующего порядка действий:

1. Создайте элемент управления.
2. Установите необходимые свойства для элемента.
3. Добавьте элемент в коллекцию формы `Controls`.

Вы можете посмотреть на код, генерируемый дизайнером форм, когда вы помещаете элементы управления на форму во время разработки (режим design), и использовать его для своих задач. В листинге 6.1 дан простой пример добавления текстового поля на форму.

**Листинг 6.1. Добавление элемента на форму**

```
// Шаг 1
 TextBox tb = new TextBox();

// Шаг 2
 tb.Location = new Point( 10, 10);
 tb.Size = new Size(100, 20);
 tb.Text = "Я был создан во время выполнения программы";

// Шаг 3
 this.Controls.Add(tb);
```

## Как пройтись по всем элементам управления на форме?

Иногда встречается задача пройтись по всем элементам управления на форме, чтобы изменить какие-либо свойства. Например, может понадобиться изменить текст у всех текстовых полей или цвет у всех кнопок. Вы можете перебрать в цикле элементы управления, используя объект `Control.ControlCollection`, возвращаемый свойством `Form.Controls`. `ControlCollection` включает в себя все элементы управления, расположенные на форме. Однако тут есть одна тонкость. Некоторые элементы управления являются контейнерами для других элементов управления. К числу таких контейнеров относятся элементы управления `GroupBox`, `Panel`, `TabPage` и др. Поэтому в таких случаях может понадобиться рекурсивный перебор всех элементов. Рассмотрим несколько примеров. Расположим на форме несколько текстовых полей и одну кнопку. Предположим, нам необходимо поменять текст у всех текстовых полей. Код для решения этой задачи приведен в листинге 6.2.

**Листинг 6.2. Получение всех элементов управления на форме**

```
private void button1_Click(object sender, EventArgs e)
{
    foreach (Control ctrl in this.Controls)
    {
        // Работаем только с текстовыми полями
        if (ctrl.GetType() == typeof(TextBox))
        ctrl.Text = "Народные советы";
    }
}
```

Если бы мы не использовали строчку

```
if (ctrl.GetType() == typeof(TextBox))
```

текст поменялся бы не только у текстовых полей, но и у кнопки. Усложним задачу. Поместим на форму элемент `Panel` и на нее поместим еще одно текстовое поле. Запустите проект и нажмите на кнопку — вы увидите, что текст в добавленном текстовом поле не изменился. Следовательно, чтобы изменить текст во всех без исключения текстовых полях, нам придется воспользоваться рекурсией (листинг 6.3).

**Листинг 6.3. Рекурсивный способ обхода всех элементов управления**

```
private void IterateControls(Control ctrl)
{
    // Работаем только с текстовыми полями
    if (ctrl.GetType() == typeof(TextBox))
    {
        ctrl.Text = "Народные советы";
    }

    // Проходим через элементы рекурсивно,
    // чтобы не пропустить элементы, которые находятся в контейнерах
    foreach (Control ctrlChild in ctrl.Controls)
    {
        IterateControls(ctrlChild);
    }
}


private void button1_Click(object sender, EventArgs e)
{
    IterateControls(this);
}
```

## Как изменить цвет границы (*Border*) у элемента управления?

Если вы хотите изменить цвет границы у своего элемента управления, то переопределите метод `OnPaint`. В листинге 6.4 приведен небольшой пример, который меняет окантовку у кнопки. Сначала мы создаем новый класс

MyButton, устанавливаем нужный цвет и добавляем кнопку на форму, используя предыдущий пример (см. листинг 6.1).

**Листинг 6.4. Изменение цвета границ у элемента управления**

```
public class MyButton : Button
{
    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        int borderWidth = 1;
        Color borderColor = Color.Green;
        ControlPaint.DrawBorder(e.Graphics, e.ClipRectangle, borderColor,
            borderWidth, ButtonBorderStyle.Solid, borderColor,
            borderWidth, ButtonBorderStyle.Solid, borderColor,
            borderWidth, ButtonBorderStyle.Solid, borderColor,
            borderWidth, ButtonBorderStyle.Solid);
    }
}

public Form1()
{
    InitializeComponent();

    MyButton btn = new MyButton();
    btn.Width = 90;
    btn.Height = 50;
    btn.Left = 100;
    btn.Top = 10;
    btn.Text = "Я новая кнопка";
    btn.Visible = true;
    this.Controls.Add(btn);
}
```

## Окантовка вокруг элемента управления

Другой вариант — обвести окантовку вокруг нужного элемента управления, используя его размеры (листинг 6.5).

**Листинг 6.5. Окантовка вокруг элемента управления**

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen p = new Pen(Color.Red, 3);


    foreach(Control ctrl in this.Controls)
    {
        if(ctrl.GetType() == typeof(CheckBox))
        {
            g.DrawRectangle(p,new Rectangle(ctrl.Location,ctrl.Size));
        }
    }
}
```

Запустив проект, вы увидите, что элемент `CheckBox` обведен красной рамочкой. Естественно, подобный прием также можно применить практически к любому элементу управления.

> ***Примечание***
>
> Примеры находятся в папке ControlsDemo на прилагаемом диске.

## Как программно перевести фокус на следующий/предыдущий (в порядке TAB) элемент управления?

Чтобы программно перевести фокус на следующий элемент управления, можно использовать метод `Control.SelectNextControl`. Предположим, мы хотим перевести фокус с текстового поля на кнопку, если в текстовом поле пользователь введет 5 символов. Тогда можно воспользоваться кодом из листинга 6.6.

**Листинг 6.6. Программный перевод фокуса на другой элемент управления**

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    const int MAX_LENGTH = 5;
```

```
    if (textBox1.Text.Length == MAX_LENGTH)
        // переводим фокус на следующий элемент управления
        SelectNextControl(textBox1,true,true,false,false);
}
```

Кстати, если у вас на форме расположено слишком много элементов управления, и вы хотите узнать порядок, по которому будет переходить фокус, то выберите в меню **View** команду **Tab Order**. Все элементы управления после данного трюка получат порядковый номер, который будет соответствовать их свойству `TabIndex` (рис. 6.1). Вы можете быстро изменить существующий порядок. Достаточно пощелкать мышкой по этим номерам в нужной последовательности.

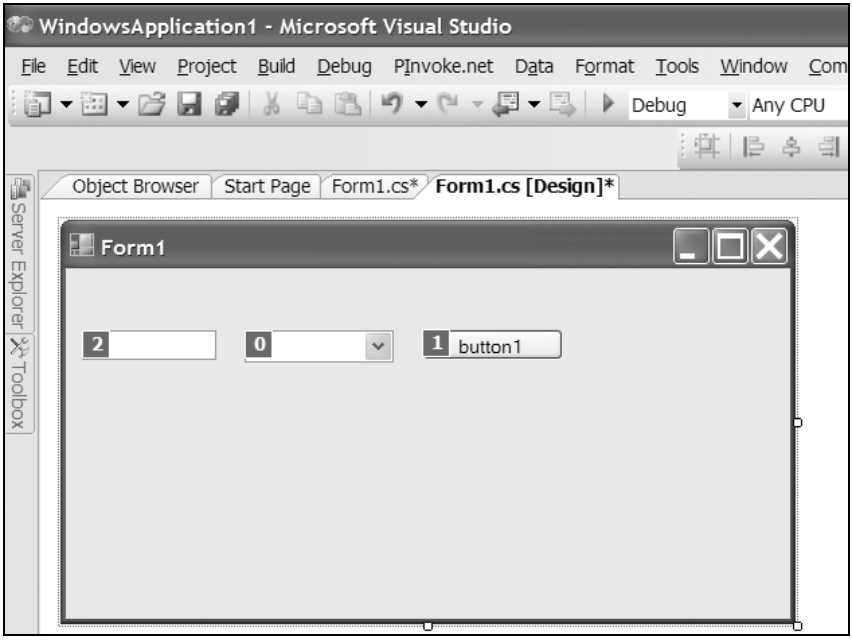



**Рис. 6.1.** Порядковые номера для `TabIndex`

## Как изменить Z-порядок элемента управления?

Метод `SetChildIndex` класса `System.Windows.Forms.Control.ControlCollection` позволяет изменять Z-порядок элемента управления на форме. Предположим, на форме одна кнопка частично перекрывает вторую

кнопку. Чтобы закрываемую кнопку вывести вперед, нужно присвоить параметру `newIndex` метода `SetChildIndex` значение 0 (листинг 6.7).

**Листинг 6.7. Изменение Z-порядка**

```
private void button2_Click(object sender, EventArgs e)
{
    // Выводим кнопку button2 на передний край
    this.Controls.SetChildIndex(button2, 0);
}
```

***ПРИМЕЧАНИЕ***

Примеры находятся в папке Controls на прилагаемом диске.

## Как узнать размеры строки в пикселах, отображаемой в каком-нибудь элементе управления?

Чтобы узнать размер строки в пикселах, нужно использовать метод `Graphics.MeasureString`. Подобная необходимость возникает, когда нужно подогнать размеры самого элемента управления под размер самой широкой строки текста. Предположим, нужно установить ширину элемента `Button` таким образом, чтобы любой длинный текст нормально отображался бы в ней (листинг 6.8).

**Листинг 6.8. Получение размеров строки в пикселах**

```
public partial class Form1 : Form
{
    const string strTextForButton = "Тестовая строкаааааааааааааа";

    public Form1()
    {
        InitializeComponent();

        button1.Text = strTextForButton;

        using (Graphics graphics = button1.CreateGraphics())
```

```
        {
            button1.Width = (int)graphics.MeasureString(strTextForButton,
                                                        Font).Width;
            this.Text = button1.Width.ToString();
        }

    }
}
```

Для сравнения закомментируйте строчки кода, начиная со слов `using (Graphics g…)`, и снова запустите проект. Вы увидите, что текст, который не может поместиться на кнопке, просто обрезается. На рис. 6.2 для наглядности представлены оба варианта.

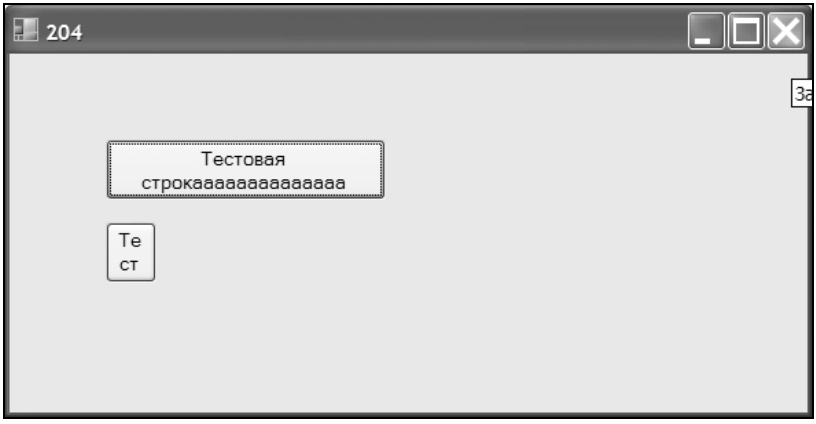



**Рис. 6.2.** Подгоняем размеры кнопки под текст

***ПРИМЕЧАНИЕ***

Пример находится в папке MeasureStringDemo на прилагаемом диске.

## Как сделать элемент управления произвольной формы?

С помощью свойства `Region` создание элемента управления произвольной формы становится тривиальной задачей. Причем не нужно создавать собст-

венных элементов управления, можно использовать готовые кнопки, метки и т. д. Возьмем, к примеру, стандартную кнопку `Button`. Пусть при нажатии на эту кнопку она примет овальную форму (листинг 6.9 и рис. 6.3).


**Рис. 6.3.** Создание овальной кнопки

**Листинг 6.9. Превращение стандартной кнопки в овальную**

```
private void button1_Click(object sender, EventArgs e)
{
    GraphicsPath gp = new GraphicsPath();
    Graphics g = CreateGraphics();

    // Создадим новый прямоугольник с размерами кнопки
    Rectangle smallRectangle = button1.ClientRectangle;

    // уменьшим размеры прямоугольника
    smallRectangle.Inflate(-3, -3);

    // создадим эллипс, используя полученные размеры
    gp.AddEllipse(smallRectangle);
    button1.Region = new Region(gp);

    // рисуем окантовоку для круглой кнопки
    g.DrawEllipse(new Pen(Color.Gray, 2),
        button1.Left + 1,
        button1.Top + 1,
```

```
        button1.Width - 3,
        button1.Height - 3);

    // освобождаем ресурсы
    g.Dispose();
}
```

***Примечание***

Проект находится в папке OvalButton на прилагаемом диске.

# Кнопки (*Button*)

## Как установить кнопку по умолчанию для формы?

Кнопкой по умолчанию (`Default`) является кнопка на форме, которая реагирует на нажатие клавиши <Enter>. На форме может быть только одна такая кнопка. Чтобы сделать любую вашу кнопку на форме кнопкой по умолчанию, присвойте свойству формы `AcceptButton` кнопку, которой нужно придать такое поведение. Это можно сделать на этапе разработки или программным способом следующим образом:

```
form1.AcceptButton = button1;
```

## Как установить кнопку отмены (*Cancel*) для формы?

Кроме кнопки по умолчанию на форме может присутствовать также и кнопка отмены, которая реагирует на нажатие клавиши <Esc>. У формы есть свойство `CancelButton`, которое отвечает за подобное поведение кнопки. Вы можете установить это свойство в дизайнере форм или программным способом:

```
form1.CancelButton = button1;
```

## Как программно вызвать событие *Click* у кнопки?

Если вы раньше программировали на Visual Basic 6.0, то у вас не было проблем с этой задачей. Достаточно было вызвать событие `Command1_Click` в

любом месте кода, чтобы выполнились операторы обработчика этого события. Но в .NET Framework этот способ не сработает. Однако решить проблему очень просто. Просто используйте метод кнопки `PerformClick`.

```
button1.PerformClick();
```

На форумах некоторыми программистами предлагаются альтернативные варианты решения. Например, можно вызвать `Buttonclick` с пустыми параметрами:

```
button1_Click(null, null);
```

Или так:

```
button1_click(button1, EventArgs.Empty);
```

## Как создать западающую кнопку?

Некоторые даже не подозревают, что элемент `CheckBox` может выглядеть как западающая кнопка. Убедитесь в этом сами. Поместите на форму элемент `CheckBox` и установите значение `Button` в его свойстве `Appearence`. Также поиграйте со значениями свойства `FlatStyle`.

> ***Примечание***
>
> Примеры действий с кнопками находятся в папке ButtonDemo на прилагаемом диске.

# Список (*ListBox*)

Простейшие операции со списком обычно не вызывают затруднений у программиста — методы `Items.Add`, `Items.Remove`, `Items.RemoveAt`, `Items.Insert`, `Items.Clear` говорят сами за себя и позволяют добавлять, удалять, вставлять и очищать список. Рассмотрим более сложные приемы.

## Автоматическая прокрутка списка

Если вам необходимо автоматически прокручивать список, чтобы всегда видеть последний добавленный элемент, то воспользуйтесь свойством `TopIndex`. Предположим, у вас имеется список, в который вы добавляете новый элемент и хотите, чтобы он был виден на экране. Тогда можно воспользоваться кодом из листинга 6.10.

**Листинг 6.10. Автоматическая прокрутка списка**

```
private void button1_Click(object sender, EventArgs e)
{
    listBox1.Items.Add(textBox1.Text);
    listBox1.TopIndex = listBox1.Items.Count - 1;
}
```

## Подгоняем ширину списка под самый длинный текст

Если необходимо, чтобы самый длинный текст был виден в списке, то нужно вычислить длину этого текста через свойство `MeasureString`, пройтись по всем элементам списка и установить новую ширину списка, используя полученные результаты.

**Листинг 6.11. Подгонка ширины списка под самый длинный текст**

```
// Подгоняем ширину списка под самый длинный текст
Graphics g = listBox1.CreateGraphics();
float maxWidth = 0f;
float height = 0f;

for (int i = 0; i < listBox1.Items.Count; ++i)
{
    float w = g.MeasureString(listBox1.Items[i].ToString(),
                              listBox1.Font).Width;
    if (w > maxWidth)
        maxWidth = w;
    height += listBox1.GetItemHeight(i);
}
g.Dispose();

listBox1.Width = (int)(maxWidth + 8 + ((height > listBox1.Height - 4) ?
                                       16 : 0)); // 16 - ширина прокрутки
```

## Как заполнить список именами файлов, перетаскиваемых из Проводника?

Операция Drag'n'Drop очень хорошо реализована в .NET Framework. Если вы хотите заполнить список именами файлов, перетаскиваемых прямо из Про-

водника, то присвойте сначала свойству `AllowDrop` списка `ListBox` значение `true` и напишите для событий `DragEnter` и `DragDrop` код, приведенный в листинге 6.12.

**Листинг 6.12. Заполнение списка именами файлов**

```
private void listBox1_DragDrop(object sender, DragEventArgs e)
{
    //Извлекаем имя перетаскиваемого файла
    string[] astrings = (string[])e.Data.GetData(DataFormats.FileDrop,
                                                 true);

    foreach (string strfile in astrings)
    {
        // только имя файла
        listBox1.Items.Add(strfile.Substring(1 +
                           strfile.LastIndexOf(@"\")));

        // или полный путь к файлу
        listBox1.Items.Add(strfile);
    }

private void listBox1_DragEnter(object sender, DragEventArgs e)
{
    // Разрешаем Drop только файлам
    e.Effect = e.Data.GetDataPresent(DataFormats.FileDrop) ?
                        DragDropEffects.All : DragDropEffects.None;
}
```

## Разделить список цветными линиями и заполнить цветным текстом

Если вам не нравится стандартный вид списка, то вы должны самостоятельно заняться отрисовкой этого элемента управления. Сначала присвойте свойству `DrawMode` значение `OwnerDrawFixed`. Тем самым вы берете на себя всю работу, связанную с отображением элемента. Далее в событии `DrawItem` вы пишете код, который будет отвечать за вывод цветного текста и цветных линий при помощи `DrawString` и `DrawLine`, как показано в листинге 6.13.

**Листинг 6.13. Вывод цветного списка**

```
private void Form1_Load(object sender, EventArgs e)
{
    listBox1.Items.Add("One");
    listBox1.Items.Add("Two");
    listBox1.Items.Add("Three");

    lstColor.Items.Add(Color.Red.Name);
    lstColor.Items.Add(Color.Yellow.Name);
    lstColor.Items.Add(Color.Green.Name);
    lstColor.Items.Add(Color.Blue.Name);
}

private void lstColor_DrawItem(object sender, DrawItemEventArgs e)
{
    if ((e.State & DrawItemState.Selected) != 0)
        e.Graphics.FillRectangle(SystemBrushes.Highlight, e.Bounds);
    else
        e.Graphics.FillRectangle(Brushes.White, e.Bounds);

    string itemText = lstColor.Items[e.Index].ToString();
    Color color = Color.FromName(itemText);

    //Рисуем строку
    e.Graphics.DrawString(itemText, Font, new SolidBrush(color),
                          e.Bounds);

    Pen pen = new Pen(color);

    //Рисуем линию под строкой
    e.Graphics.DrawLine(pen, e.Bounds.X, e.Bounds.Bottom - 1,
                        e.Bounds.Right, e.Bounds.Bottom - 1);
 }
```

***ПРИМЕЧАНИЕ***

Примеры работы с `ListBox` находятся в папке ListBoxDemo на прилагаемом диске.

# Поле со списком (*ComboBox*)

## Подгоняем ширину поля со списком под самый длинный текст

Мы уже рассматривали пример подгонки ширины списка `ListBox` под самый длинный текст элемента. Такую же технику (листинг 6.14) можно применять и для поля со списком.

**Листинг 6.14. Подгонка ширины комбинированного окна под самый длинный текст**

```
private void button2_Click(object sender, EventArgs e)
{
    Graphics g = comboBox1.CreateGraphics();
    float maxWidth = 0f;
    foreach (object o in comboBox1.Items)
    {
        float w = g.MeasureString(o.ToString(), comboBox1.Font).Width;
        if (w > maxWidth)
            maxWidth = w;
    }
    g.Dispose();
    // 28 - учитываем ширину кнопки в поле со списком
    comboBox1.Width = (int)maxWidth + 28;
}
```

## Поддержка автозавершения

Обратите внимание на новинку .NET Framework 2.0, относящуюся к полю со списком: теперь `ComboBox` поддерживает автозавершение. Для реализации этой функциональности вам надо задействовать три свойства. Первое — установите стиль автозавершения в свойстве `AutoCompleteMode`, которое может принимать одно из следующих значений:

- `None` — автозавершение отключено (по умолчанию);
- `Suggest` — используются значения из раскрывающегося списка;
- `Append` — значения дополняются при вводе;
- `SuggestAppend` — комбинация `Suggest` и `Append`.

Если вы выбрали любое значение, отличное от `None`, то в свойстве `AutoCompleteSource` нужно указать имя системного источника данных для автозавершения. Здесь допустимы следующие значения:

- `FileSystem` — пользователю предлагаются файлы и папки текущей системы, например, если вы наберете `c:\wi`, то будет предложено c:\windows;
- `HistoryList` — пользователю предлагается выбрать из системного списка истории посещений адрес Uniform Resource Locators (URL);
- `RecentlyUserList` — пользователю предлагаются на выбор адреса из списка недавно использованных. Это не обязательно должен быть адрес веб-страницы, могут использоваться, например, команды, вводимые в диалоговом окне **Выполнить**;
- `AllUrl` — объединение предыдущих двух свойств `HistoryList` и `RecentlyUserList`;
- `AllSystemSources` — все варианты из `AllUrl` и `FileSystem`;
- `CustomSource` — собственный набор вариантов, определяемый разработчиком;
- `None` — ввод осуществляется пользователем вручную;
- `ListItems` — выбирается один из вариантов, перечисленных в коллекции `ComboBox.ObjectCollection`.

Если вы выберете значение `CustomSource`, то `ComboBox` ожидает данные для автозавершения в виде набора элементов, указанных в свойстве `AutoCompleteCustomSource`. Обратитесь к документации за более подробной информацией по данному вопросу. Кстати, текстовые поля также поддерживают автозавершение.

## Как раскрыть поле со списком программным способом?

Как ни странно, решается эта задача удивительно просто: достаточно установить в значение `true` свойство `ComboBox.DroppedDown`:

```
comboBox1.DroppedDown = true;
```

## Как запретить раскрытие списка?

Теперь нас интересует другая задача — запретить пользователю раскрывать поле со списком с помощью мыши. Для этого нужно переопределить метод `WndProc`, как показано в листинге 6.15.

**Листинг 6.15. Запрет на раскрытие списка**

```
protected override void WndProc(ref System.Windows.Forms.Message m)
{
    // константы для левой кнопки мыши
    const int WM_LBUTTONDOWN = 0x201;
    const int WM_LBUTTONDBLCLK = 0x203;

    if (m.Msg == WM_LBUTTONDOWN || m.Msg == WM_LBUTTONDBLCLK)
        return;
    base.WndProc(ref m);
}

public Form1()
{
    InitializeComponent();

    mycomboBox cboNotDrop = new mycomboBox();
    cboNotDrop.Parent = this;
    cboNotDrop.Width = 140;
    cboNotDrop.Height = 60;
    cboNotDrop.Items.Add("One");
    cboNotDrop.Items.Add("Two");
    // ... и так далее
}
```

Запустите проект и попробуйте при помощи мыши раскрыть список у `ComboBox` — у вас ничего не получится. Переведите фокус на этот элемент и попробуйте раскрыть его с помощью клавиатуры.

## Как изменить высоту элементов списка у элемента управления *ComboBox?*

Если вас не устраивает высота элементов списка, то можно изменить его, установив шрифт нужной высоты (листинг 6.16). Результат показан на рис. 6.4.

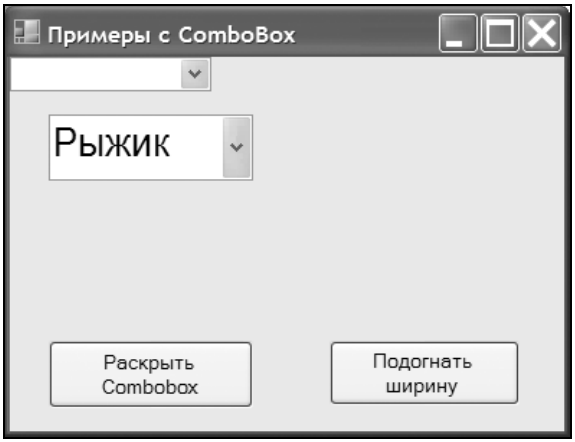



**Рис. 6.4.** Увеличиваем высоту элементов списка

**Листинг 6.16. Изменение высоты элементов списка**

```
comboBox1.Font = new Font("Arial", 16);
```

Если вам нужно увеличить высоту строки, отводимой под элемент списка, не трогая размеры шрифта, то используйте в этом случае сообщение `CB_SETITEMHEIGHT`, передав в параметре `wParam` значение `-1`, а в параметре `lParam` требуемую высоту (листинг 6.17). Получится список, показанный на рис. 6.5.

**Листинг 6.17. Увеличение высоты под элемент списка**

```
const int CB_SETITEMHEIGHT = 0x0153;

[DllImport("user32.dll")]
public static extern IntPtr SendMessage(IntPtr hWnd,
UInt32 Msg, Int32 wParam, UInt32 lParam);
```

```
// Устанавливаем желаемую высоту
SendMessage(comboBox1.Handle, CB_SETITEMHEIGHT, -1, 40);
```

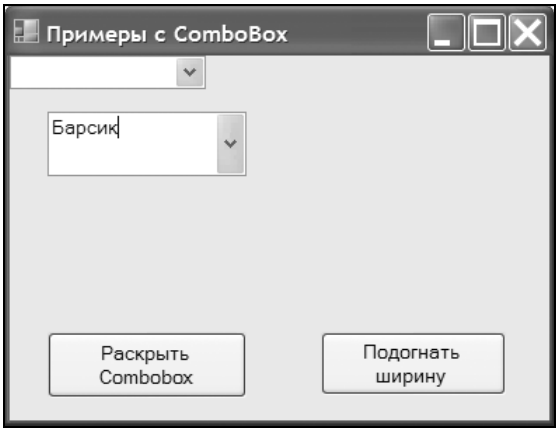



**Рис. 6.5.** Изменение высоты, отводимой под элемент списка `ComboBox`

## Как установить желаемую высоту выпадающего списка у *ComboBox?*

У поля со списком имеется свойство `MaxDropDownItems`, которое определяет число видимых элементов в выпадающем списке. Таким образом, вам остается выбрать нужное значение данного свойства, чтобы контролировать высоту этого списка.

## Как использовать *ComboBox* для редактирования данных в *ListView?*

В завершение рассказа об элементе управления `ComboBox` хочу порекомендовать почитать статью из Базы Знаний Microsoft "How to use a ComboBox control to edit data in a ListView control in Visual C#" (на англ. языке), с которой вы можете ознакомиться на странице **http://support.microsoft.com/kb/320344/**. В этой статье рассказывается, как можно использовать поле со списком `ComboBox` для редактирования данных в элементе управления `ListView`.

***Примечание***

Примеры работы с `ComboBox` находятся в папке ComboBoxDemo на прилагаемом диске.

# Текстовые поля (*TextBox*)

## Подсчет числа строк в многострочном текстовом поле

У многострочного текстового поля (`Multiline == true`) свойство `Lines` возвращает массив строк в текстовом поле без учета переноса слов, которые не умещаются по ширине поля. Поэтому, если установить у свойства `WordWrap` значение `true`, то свойство `Lines` не сможет правильно посчитать число строк текста, так как учитывает только строки, разделенные специальным символом переноса строки (возврат каретки). В этом случае можно воспользоваться сообщением `EM_GETLINESCOUNT`. Рассмотрим на примере (листинг 6.18), как именно это сделать. Создадим текстовое поле и поместим туда какую-нибудь длинную строку, а также сразу установим значения свойств `Multiline` и `WordWrap` в `true`. Также поместим на форму кнопку, с помощью которой пользователь сможет узнать число строк в текстовом поле.

**Листинг 6.18. Подсчет строк в многострочном текстовом поле**

```
using System.Runtime.InteropServices;

[DllImport("user32.dll")]
public static extern int SendMessage(IntPtr hWnd,
    UInt32 Msg, UInt32 wParam, UInt32 lParam);

const int EM_GETLINECOUNT = 0x00BA;

private void Form1_Load(object sender, EventArgs e)
{
    txtLinesCount.Multiline = true;
    txtLinesCount.WordWrap = true;
    txtLinesCount.Height = 98;
    txtLinesCount.Text += "У Лукоморья дуб зеленый; ";
    txtLinesCount.Text += "Златая цепь на дубе том: "
    txtLinesCount.Text += "И днем и ночью кот ученый ";
```

```
    txtLinesCount.Text += "Все ходит по цепи кругом";
}

private void butCount_Click(object sender, EventArgs e)
{
    int LineCount;
    LineCount = SendMessage(txtLinesCount.Handle, EM_GETLINECOUNT, 0, 0);
    MessageBox.Show("Число строк: " + LineCount);
}
```

## Фильтрация заданных символов при вводе с клавиатуры

Можно не допустить ввод нежелательных символов с клавиатуры, обрабатывая событие `KeyPress`. Для этого нужно обработать нажатую клавишу при помощи свойства `KeyChar` класса `KeyPressEventArgs`. Например, мы не хотим, чтобы пользователь мог вводить символ "Б" и нажимать на клавишу <Enter>. В этом случае подойдет код, приведенный в листинге 6.19.

**Листинг 6.19. Фильтруем вводимые символы**

```
private void textBox1_KeyPress(object sender,
                               System.Windows.Forms.KeyPressEventArgs e)
{
     if(e.KeyChar == (char)13 || e.KeyChar == 'Б')
          e.Handled = true;
}
```

Немного модифицируем пример, чтобы позволить вводить в текстовое поле только цифры (листинг 6.20).

**Листинг 6.20. Разрешение на ввод только цифр**

```
if (! Char.IsDigit(e.KeyChar) )
         e.Handled = true;
```

Еще улучшим наш пример. Допустим, мы хотим произвести фильтрацию ввода данных по определенному шаблону. Для конкретности предположим, что мы хотим разрешить пользователю вводить не только цифры, но и знаки

плюс, минус и запятую. В этом случае мы создаем шаблон разрешенных символов и при вводе проверяем, входит ли данный символ в заданный шаблон (листинг 6.21).

**Листинг 6.21. Ввод символов по заданному шаблону**

```
// по шаблону разрешаем вводить в поле цифры, знаки плюс, минус и запятую
string pattern = "0123456789+-,";

if(!Char.IsControl(e.KeyChar))
  if (pattern.IndexOf(e.KeyChar.ToString()) < 0)
     e.Handled = true;
```

## Как заблокировать контекстное меню в текстовом поле?

Если вам необходимо заблокировать для пользователя возможность воспользоваться стандартным контекстным меню, то просто создайте пустое контекстное меню следующим образом:

```
textBox1.ContextMenu = new ContextMenu();
```

Надо отметить, что при этом буфер обмена Windows по-прежнему работает, и опытный пользователь может выполнить стандартные операции копирования, вставки текста с помощью быстрых клавиш.

## Запрет вставки текста из буфера обмена Windows

Мы знаем, как заблокировать контекстное меню у текстового поля, чтобы пользователь не мог редактировать текст. Но, как же все-таки запретить вставку текста из буфера обмена? Вставка из буфера обмена осуществляется при помощи сообщения `WM_PASTE`, поэтому нам надо отловить это сообщение в процедуре `WndProc` и отменить его (листинг 6.22).

**Листинг 6.22. Запрет на вставку из буфера обмена**

```
class TextBoxEx : TextBox
{
    const int WM_PASTE = 0x0302;
```

```
    protected override void WndProc(ref Message m)
    {
        //Запрещаем обрабатывать WM_PASTE
        if (m.Msg == WM_PASTE)
            return;
        base.WndProc(ref m);
    }
}

public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        TextBoxEx txtNotPaste = new TextBoxEx();
        txtNotPaste.Parent = this;
        txtNotPaste.Multiline = true;
        txtNotPaste.Height = 60;
        // ... и так далее
    }
```

Как видите, мы выполняем наследование от стандартного класса `TextBox` и запрещаем для унаследованного класса вставку из буфера обмена.

## Как ввести многострочный текст в текстовое поле программно?

Иногда требуется ввести в текстовое поле текст, расположенный в несколько строк. Во-первых, можно установить свойство `Multiline` в значение `true` и далее воспользоваться свойством `Lines`, которое должно содержать массив необходимых строк (листинг 6.23).

**Листинг 6.23. Создание многострочного текста программным способом**

```
string[] strWeekDay = { "Понедельник", "Вторник", "Среда" };
txtLinesCount.Multiline = true;
txtLinesCount.Lines = strWeekDay;
```

А можно воспользоваться комбинацией двух escape-последовательностей `"\r\n"`, как показано в листинге 6.24.

**Листинг 6.24. Создание многострочного текста при помощи escape-последовательностей**

```
// Используем escape-последовательности для переноса текста
// на новую строку
txtLinesCount.Text = "Раз\r\nДва\r\nТри";
```

В .NET Framework определено специальное свойство `System.Environment.NewLine`, которое является аналогом комбинации этих последовательностей. Использование этого свойства предпочтительнее, так как код становится более платформонезависимым (листинг 6.25).

**Листинг 6.25. Третий способ создания многострочного текста**

```
// Используем свойство NewLine для переноса строк
txtLinesCount.Text = "Месяцы года:" + Environment.NewLine + "Декабрь" +
                     Environment.NewLine + "Январь...";
```

## Как сделать так, чтобы символы вводились в нужном регистре?

Я помню, что для решения этой задачи на Visual Basic 5.0 мне приходилось писать собственную процедуру, которая занималась конвертацией вводимых символов в нужный регистр. Теперь у текстовых полей есть свойство `CharacterCasing`, которое позволяет решить эту проблему одной строчкой кода (листинг 6.26).

**Листинг 6.26. Ввод символов в верхнем регистре**

```
// Все вводимые символы переводятся в верхний регистр
this.textBox1.CharacterCasing =
                     System.Windows.Forms.CharacterCasing.Upper;
```

## Как избавиться от звукового сигнала при нажатии на клавишу ввода?

Когда вы нажимаете на клавишу <Enter> при наборе текста в текстовом поле, то слышен щелчок, издаваемый системным динамиком. Если вы хотите изба-

виться от этого звука, то вам необходимо перехватить нажатие этой клавиши, как делается в листинге 6.27.

**Листинг 6.27. Избавление от звукового сигнала при нажатии на клавишу <Enter>**

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == (char)13)
        e.Handled = true;
    else
        base.OnKeyPress(e);
}
```

В листинге 6.27 показан пример для отдельного текстового поля. Если вы хотите применить этот способ ко многим текстовым полям, то проще определить новый класс наследованием от `TextBox` и переопределить событие `OnKeyPress` (листинг 6.28).

**Листинг 6.28. Переопределение события `OnKeyPress`**

```
public class MyTextBox : TextBox
{
     protected override void OnKeyPress(KeyPressEventArgs e)
     {
          if(e.KeyChar == (char) 13)
               e.Handled = true;
          else
               base.OnKeyPress (e);
     }
}
```

## Как выделить текст программным способом?

Например, если в вашей программе осуществляется поиск определенной строки в текстовом поле, то выделение найденного текста привлечет внимание пользователя. Чтобы выделить текст программными средствами, нужно присвоить свойству `SelectionStart` значение, соответствующее начальной позиции текста, который вам требуется выделить. При этом крайняя левая позиция текста имеет значение 0. Если значение свойства `SelectionStart` больше или равно числу знаков в текстовом поле, то курсор помещается за

последним знаком. Если нужно выделить весь текст, то присвойте свойству `SelectionLength` значение, равное длине текста, который требуется выделить. Если значение `SelectionLength` больше нуля, то выделяется указанное количество знаков, начиная с текущей позиции курсора. Пример, приведенный в листинге 6.29, выделяет весь текст в поле `textBox1`.

**Листинг 6.29. Выделение текста**

```
// Выделяем весь текст в текстовом поле
private void textBox1_Enter(object sender, System.EventArgs e)
{
   textBox1.SelectionStart = 0;
   textBox1.SelectionLength = textBox1.Text.Length;
}
```

***ПРИМЕЧАНИЕ***

Примеры работы с текстовыми полями находятся в папке TextBoxDemo на прилагаемом диске.

# Элемент *RichTextBox*

## Просмотр форматированного текста RTF

Не все знают, что форматированный текст RTF состоит из определенных команд управления, которые формируют текст в нужном виде. Вы можете посмотреть, как выглядит ваш текст, отформатированный определенным образом в текстовом поле `RichTextBox`, при помощи свойства `Rtf`. Поместите на форму элемент `RichTextBox`. Запустите проект, скопируйте какой-нибудь кусочек текста из файла DOC или RTF и вставьте его в подготовленное текстовое поле. Теперь, чтобы увидеть команды управления, содержащиеся в тексте (рис. 6.6), достаточно одной строки кода, приведенной в листинге 6.30.

**Листинг 6.30. Просмотр команд управления RTF**

```
private void butGetRTF_Click(object sender, EventArgs e)
{
    textBox1.Text = richTextBox1.Rtf;
}
```

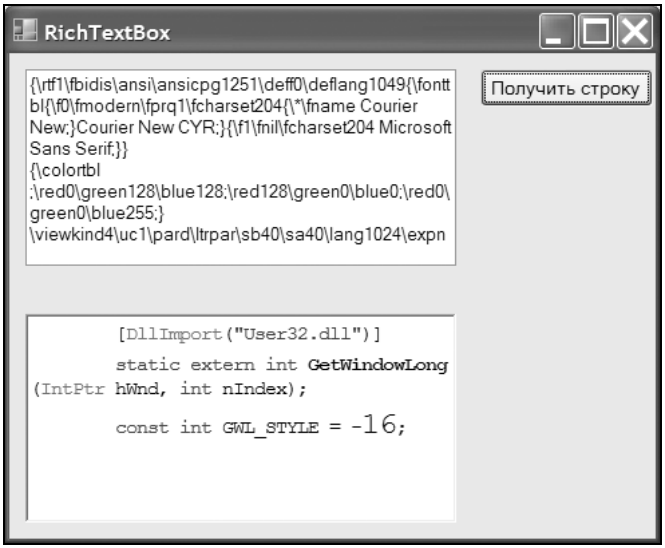



**Рис. 6.6.** Команды управления RTF

## Как управлять цветом и шрифтами в *RichTextBox?*

Чтобы форматировать текст, изменяя его цвет или шрифт, нужно выделить его и далее применять свойства `SelectionFont` и `SelectionColor`. Не забудьте перед изменениями установить фокус на элементе, иначе ваш код пропадет впустую. Если никакой текст не был выделен, то новый текст, который будет вводиться с клавиатуры или вставляться, будет иметь полужирный шрифт красного цвета. Соответствующий код приведен в листинге 6.31.

**Листинг 6.31. Управление цветом и шрифтом в `RichTextBox`**

```
private void butFormatRTF_Click(object sender, EventArgs e)
{
    // Устанавливаем фокус на richTextBox
    richTextBox1.Focus();
    // Устанавливаем цвет для выделенного текста
    richTextBox1.SelectionColor = Color.Red;
    // Устанавливаем шрифт
    richTextBox1.SelectionFont = new Font("Courier", 10, FontStyle.Bold);
}
```

## Как управлять текстом-гиперссылкой в *RichTextBox?*

По умолчанию элемент управления `RichTextBox` имеет свойство `DetectUrls`, равное `true`. Это означает, что текст, представляющий собой URL, автоматически преобразуется в гиперссылку, как это мы обычно привыкли видеть в браузерах. Слово-гиперссылка подчеркивается линией, а мышь при подведении к такой ссылке приобретает вид руки с вытянутым указательным пальцем. Но этого недостаточно, чтобы щелчок на ссылке запустил указанную страницу. Вам необходимо написать свой код (листинг 6.32) для события `LinkClicked`.

**Листинг 6.32. Управление гиперссылкой в `RichTextBox`**

```
private void richTextBox1_LinkClicked(object sender,
                                      LinkClickedEventArgs e)
{
    System.Diagnostics.Process.Start(e.LinkText);
}
```

## Поддержка Drag'n'Drop

Вы знаете, что в документе Word можно перетаскивать выделенный текст с помощью мыши. Вы можете обеспечить подобную функциональность в своем проекте. Установите свойство `AllowDrop` в значение `true` и добавьте свой код (листинг 6.33) для обработки событий `DragEnter` и `DragDrop`.

**Листинг 6.33. Поддержка Drag'n'Drop**

```
public Form1()
{
    InitializeComponent();
    richTextBox1.AllowDrop = true;
    this.richTextBox1.DragEnter +=
        new System.Windows.Forms.DragEventHandler(
                                    this.richTextBox1_DragEnter);
    this.richTextBox1.DragDrop +=
        new System.Windows.Forms.DragEventHandler(
                                    this.richTextBox1_DragEnter);
}
```

```
private void richTextBox1_DragEnter(object sender,
                                    System.Windows.Forms.DragEventArgs e)
{
    if (((DragEventArgs)e).Data.GetDataPresent(DataFormats.Text))
        ((DragEventArgs)e).Effect = DragDropEffects.Move;
    else
        ((DragEventArgs)e).Effect = DragDropEffects.None;
}

private void richTextBox1_DragDrop(object sender, DragEventArgs e)
{
    richTextBox1.LoadFile((String)e.Data.GetData("Text"),
        System.Windows.Forms.RichTextBoxStreamType.RichText);
}
```

## Как определить наличие полос прокрутки в элементе *RichTextBox?*

Проверить наличие полос прокрутки в элементе `RichTextBox` можно с помощью стиля `WS_HSCROLL` (горизонтальная полоса прокрутки) и `WS_VSCROLL` (вертикальная полоса прокрутки). Проверка (листинг 6.34) осуществляется при помощи функции Windows API `GetWindowLong`.

**Листинг 6.34. Проверка на наличие полосы прокрутки в `RichTextBox`**

```
[DllImport("User32.dll")]
static extern int GetWindowLong(IntPtr hWnd, int nIndex);
const int GWL_STYLE = -16;
const int WS_HSCROLL = 0x00100000;
const int WS_VSCROLL = 0x00200000;

// Проверка на наличие вертикальной прокрутки
bool IsVertScrollPresent(Control control)
{
    int style = GetWindowLong(control.Handle, GWL_STYLE);
    return (style & WS_VSCROLL) > 0;
}

// Проверка на наличие горизонтальной прокрутки
bool IsHorScrollPresent(Control control)
```

```
{
    int style = GetWindowLong(control.Handle, GWL_STYLE);
    return (style & WS_HSCROLL) > 0;
}

private void butScrollExist_Click(object sender, EventArgs e)
{
    textBox1.Text ="Вертикальная прокрутка: " +
        IsVertScrollPresent(richTextBox1).ToString() +
        Environment.NewLine + "Горизонтальная прокрутка: " +
        IsHorScrollPresent(richTextBox1).ToString();
}
```

В принципе, этот код можно применить также для текстовых полей и других элементов, которые имеют встроенные полосы прокрутки.

## Как запретить вставку

Вставлять из буфера обмена в `RichTextBox` можно двумя способами: через комбинацию клавиш <Ctrl>+<V> или через комбинацию клавиш <Shift>+ +<Insert>. Таким образом, если вам необходимо установить запрет на вставку из буфера обмена, то нужно перехватить нажатия на эти клавиши, как это сделано в листинге 6.35.

**Листинг 6.35. Запрет на вставку из буфера обмена**

```
public Form1()
{
    RichTextBoxEx rboxex = new RichTextBoxEx();
    rboxex.Parent = this;
    rboxex.Top = 400;
}

class RichTextBoxEx : RichTextBox
{
    protected override bool ProcessCmdKey(ref Message msg, Keys keyData)
    {
        //Проверяем на нажатие Shift+Insert и Ctrl+V
        if ((keyData & (Keys.Shift | Keys.Insert)) ==
                       (Keys.Shift | Keys.Insert)
            || ((keyData & (Keys.Control | Keys.V)) ==
                           (Keys.Control | Keys.V)))
```

```
            return true;
        return base.ProcessCmdKey(ref msg, keyData);
    }
}
```

Мы создали новый класс `RichTextBoxEx` и в методе `ProcessCmdKey` отслеживаем нажатия интересующих нас клавиш, не позволяя вставлять содержимое буфера обмена в наш элемент управления.

Более подробную информацию о формате RTF вы можете прочитать в Википедии по адресу **http://ru.wikipedia.org/wiki/Rich_Text_Format**.

> ***Примечание***
>
> Примеры работы с элементом `RichTextBox` находятся в папке RichTextBoxDemo на прилагаемом диске.

# Элемент управления *MaskedTextBox*

В .NET Framework 2.0 появился новый элемент управления `MaskedTexBox`, который служит для ввода данных определенного формата. Символы маски нужно задать во время разработки, чтобы указать тип и формат данных, которые может ввести пользователь. Например, можно настроить элемент таким образом, чтобы пользователь мог вводить только цифры, цифры и скобки (например, телефонный номер (095)6015629), цифры и тире (например, номер страховки 45-4993-3459) и другие варианты. Рекомендую самостоятельно изучить возможности элемента и почитать документацию. Появление данного элемента избавило программистов от необходимости создания собственных элементов на основе текстового поля для решения подобных задач.

# Элемент *DateTimePicker*

## Как показать пустой текст, если в *DateTimePicker* не выбрана дата?

Если по каким-то причинам вам нужно показать пустой текст в элементе `DateTimePicker`, то используйте код, приведенный в листинге 6.36, в обработчике события `Click` кнопки.

**Листинг 6.36. Вывод пустого текста в `DateTimePicker`**

```
dateTimePicker1.CustomFormat=" ";
dateTimePicker1.Format=DateTimePickerFormat.Custom;
```

## Как программно раскрыть *DateTimePicker?*

Элемент `DateTimePicker` можно раскрыть при помощи клавиши <F4> (когда он имеет фокус ввода). Поэтому можно просто послать соответствующую команду, как показано в листинге 6.37.

**Листинг 6.37. Программное раскрытие списка `DateTimePicker`**

```
// Устанавливаем фокус
dateTimePicker.Focus();
// Посылаем команду
SendKeys.Send("{F4}");
```

***Примечание***

Примеры работы с `DateTimePicker` находятся в папке DateTimePickerDemo на прилагаемом диске.

# Элементы *Label* и *Panel*

## Полупрозрачная надпись

Если вы будете выбирать цвет для элемента `Label` при помощи диалогового окна **ColorPicker**, то вам не удастся сделать элемент полупрозрачным. Но можно обмануть редактор свойств. Предположим, мы хотим видеть розовую полупрозрачную надпись. Выбираем подходящий цвет в **ColorPicker** через вкладку **Custom**. После выбора в текстовом поле редактора появится выбранное значение, состоящее из трех чисел, например, 255; 192; 192. Но цвет может еще иметь альфа-канал прозрачности. Вручную введем значение для альфа-канала в начале строки и получим, например 60; 255; 192; 192. Подтвердите ввод и вы увидите, что надпись на экране стала полупрозрачной. Естественно, можно добиться такого же результата и программным путем (листинг 6.38).

**Листинг 6.38. Создание полупрозрачной надписи и панели**

```
label1.BackColor = Color.FromArgb(60, 255, 192, 192);
```

Как выглядит полупрозрачная подпись, показано на рис. 6.7.

Этот прием сработает и для элемента управления `Panel`.

***Примечание***

Пример полупрозначных элементов `Label` находится в папке LabelAndPanel на прилагаемом диске.


**Рис. 6.7.** Полупрозрачная надпись на фоне фотографии

## Использование *Label* в виде разделительной линии как элемент дизайна

Есть очень любопытный прием использования элемента управления `Label` не по прямому назначению. Положите на форму метку `Label`, уберите текст из

свойства `Text`, присвойте свойству `AutoSize` значение `false`, свойству `BorderStyle` значение `Fixed3D` и установите высоту элемента в 2 пиксела. В вашем распоряжении появился очень симпатичный элемент дизайна, который можно использовать в качестве разделительной линии (рис. 6.8).

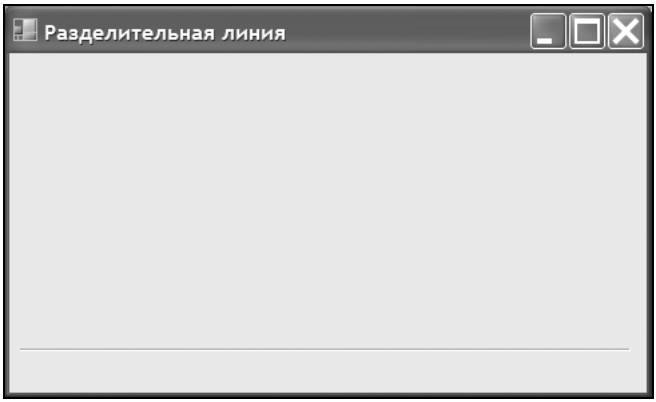



**Рис. 6.8.** Использование `Label` в качестве разделительной линии

***ПРИМЕЧАНИЕ***

Пример с разделительной линией находится в папке LabelLine на прилагаемом диске.

# Элемент *LinkLabel*

Элемент управления `LinkLabel` позволяет создавать гиперссылки на удаленные или локальные ресурсы. `LinkLabel` является потомком элемента управления `Label`, и поэтому многие программисты считают его простым и неинтересным элементом управления, который позволяет создавать простую гиперссылку, как на веб-странице. Но если это было бы так, то вряд ли Microsoft стала бы добавлять такой примитивный элемент на панель инструментов Visual Studio. На самом деле возможности элемента гораздо шире, чем представляется на первый взгляд, и даже документация не отражает всю мощь и многогранность `LinkLabel`. Кроме стандартных свойств, которые присущи обычному `Label`, `LinkLabel` также имеет ряд своих собственных свойств: `ActiveLinkColor`, `DisabledColor`, `LinkArea`, `LinkBehavior`,

LinkColor, Links, LinkVisited, VisitedLinkColor. Надо помнить, что элемент управления LinkLabel не создает ссылку автоматически и не изменяет цвет ссылки после ее посещения (по умолчанию), вы сами должны позаботиться об этой функциональности.

## Отображение лишь части текста в виде ссылки

Не всегда требуется использовать весь текст в виде ссылки. Предположим, вы используете текст следующего вида: "На нашем сайте вы найдете дополнительную информацию". Ссылкой в данном тексте является только словосочетание "нашем сайте". Для реализации этой задачи используется свойство LinkArea, которое определяет символы текста на элементе LinkLabel в виде ссылки. Свойство Start структуры LinkArea задает индекс символа строки из свойства Text, который будет соответствовать началу ссылки (нумерация начинается с 0). Свойство Length структуры LinkArea задает число символов в ссылке, включая начальный. Таким образом, вам нужно разместить на форме элемент LinkLabel, присвоить свойству Text значение "На нашем сайте вы найдете дополнительную информацию" и установить для свойства LinkArea значение (3, 11). Это можно сделать на этапе разработки в дизайнере форм или программным способом, приведенным в листинге 6.39.

**Листинг 6.39. Установка свойств для LinkLabel**

```
lnkMySite.Text = "На нашем сайте вы найдете дополнительную информацию"
lnkMySite.LinkArea = new LinkArea(3, 11);
```

Осталось только написать обработчик события LinkClicked для обработки нажатия на ссылку (листинг 6.40).

**Листинг 6.40. Обработчик события LinkClicked**

```
private void lnkMySite_LinkClicked(object sender,
                                   LinkLabelLinkClickedEventArgs e)
{
    System.Diagnostics.Process.Start("http://rusproject.narod.ru/");
}
```

Кстати, в этом коде присутствует еще одна хитрость. В самом элементе LinkLabel вы не найдете упоминания на ссылку **http://rusproject.narod.ru/**,

но мы используем этот адрес в обработчике события `LinkClicked` через метод `Start` класса `System.Diagnostics.Process`. Этот метод запустит на компьютере пользователя браузер по умолчанию.

## Несколько ссылок в одном *LinkLabel*

Рассмотрим еще один момент. Вы можете поместить на форму элемент `LinkLabel`, который будет содержать несколько ссылок, а также написать один общий обработчик события для всех ссылок. Итак, поместите на форму новый элемент `LinkLabel`, который будет содержать строку из названий популярных поисковых систем Yandex, Google, Rambler и GoGo. Обратите внимание, что у `LinkLabel` имеется свойство `LinkBehavior`, которое позволяет управлять внешним видом ссылок. Например, мы хотим, чтобы подчеркивание появлялось при наведении на ссылку мышкой — тогда используем свойство `LinkBehavior.HoverUnderline`. Формируем текст, состоящий из названий поисковых порталов, и затем добавляем ссылки на эти службы в коллекцию `Links`. Осталось только написать общий обработчик события (листинг 6.41), и пример готов.

**Листинг 6.41. Управление ссылками в `LinkLabel`**

```
private void Form1_Load(object sender, EventArgs e)
{
    lnkSearchUrl.Text = "Yandex Google Rambler GoGo";
    lnkSearchUrl.LinkBehavior = LinkBehavior.HoverUnderline;
    lnkSearchUrl.Links.Add(0, 6, "www.yandex.ru");
    lnkSearchUrl.Links.Add(7, 6, "www.google.ru");
    lnkSearchUrl.Links.Add(14, 7, "www.rambler.ru");
    lnkSearchUrl.Links.Add(22, 4, "www.gogo.ru");
    lnkSearchUrl.LinkClicked +=
        new LinkLabelLinkClickedEventHandler(lnkCommon_LinkClicked);
}

private void lnkCommon_LinkClicked(object sender,
                                   LinkLabelLinkClickedEventArgs e)
{
    LinkLabel lnk = new LinkLabel();
    lnk = (LinkLabel)sender;
    lnk.Links[lnk.Links.IndexOf(e.Link)].Visited = true;
    System.Diagnostics.Process.Start(e.Link.LinkData.ToString());
}
```

***Примечание***

Пример с `LinkLabel` находится в папке LinkLabelDemo на прилагаемом диске.

# *NotifyIcon* — значок в области уведомлений

В свое время вопрос, как добавить значок к часикам, был самым популярным на форумах, на которых обсуждались языки программирования. Для реализации этой задачи требовалось большое количество кода с привлечением функций Windows API. Существовали даже платные библиотеки, которые позволяли программисту решить эту проблему. Но с выходом .NET Framework это стало тривиальной задачей. Проведем небольшой ликбез для начинающих программистов. Вы замечали, что многие программы — брандмауэры, антивирусы и интернет-пейджеры — работают в фоновом режиме, не занимая места на панели задач. Рассмотрим пример, в котором наша программа будет сворачиваться в область уведомлений, появляться на экране при двойном щелчке по значку, а также создадим контекстное меню для значка. Запустите новый проект, возьмите на панели инструментов элемент управления `NotifyIcon` и расположите его на форме. Элемент под именем `notifyIcon1` появится в специальном контейнере под формой, так как не имеет визуального представления на форме. Установите в свойстве `Text` любой текст, который будет появляться при наведении указателя мыши на значок. Для свойства `Icon` установите необходимый значок, который будет виден в области уведомлений. Учтите, что в Express-версиях Visual Studio нельзя создавать собственные значки, но вы можете найти в Интернете программы для создания значков (в том числе и бесплатные). Перейдем к кодированию. Добавьте обработчик события для события формы `Resize`, который будет прятать приложение при сворачивании (листинг 6.42). В этом случае ваша программа не будет видна на панели задач.

**Листинг 6.42. Прячем приложение с панели задач**

```
private void Form1_Resize(object sender, System.EventArgs e)
{
   if (FormWindowState.Minimized == WindowState)
      Hide();
}
```

Теперь необходимо добавить обработчик события для события NotifyIcon.DoubleClick, в котором мы напишем код, восстанавливающий окно программы при двойном щелчке мыши (листинг 6.43).

**Листинг 6.43. Обработка двойного щелчка**

```
private void notifyIcon1_DoubleClick(object sender,
                                     System.EventArgs e)
{
    Show();
    WindowState = FormWindowState.Normal;
}
```

Наше приложение практически готово к работе. Значок появляется в области уведомлений, не появляется на панели задач при сворачивании и появляется главное окно программы при двойном щелчке. Осталось только добавить контекстное меню для значка. Для этого с панели инструментов берем элемент ContextMenu (или новый элемент ContextMenuStrip, который появился в .NET Framework 2.0) и добавляем его на форму. Создаем меню стандартным образом и для каждой созданной команды меню пишем свой обработчик события. После этого осталось в свойстве ContextMenu элемента NotifyIcon установить связь с созданным контекстным меню из выпадающего списка. Ваше приложение готово.

## Как создать мигающий значок в области уведомлений?

Вы, вероятно, встречали программы, которые мигают своим значком в области уведомлений, привлекая к себе ваше внимание. Например, программа сигнализирует о появлении новых писем или о другом событии. Мы доработаем предыдущую программу таким образом, чтобы наш значок тоже умел мигать. Делается это очень просто (листинг 6.44). Используя таймер, мы управляем свойством Visible элемента NotifyIcon. Итак, добавляем в проект элемент Timer и устанавливаем для свойства Interval значение 1000. Добавим также на форму кнопку, которая будет запускать таймер.

**Листинг 6.44. Создание мигающего значка**

```
private void button1_Click(object sender, EventArgs e)
{
```

```
    this.timer1.Start();
}

private void timer1_Tick(object sender, EventArgs e)
{
    if (notifyIcon1.Visible)
        notifyIcon1.Visible = false;
    else
        notifyIcon1.Visible = true;
}

private void notifyIcon1_Click(object sender, EventArgs e)
{
    this.timer1.Stop();
    notifyIcon1.Visible = true;
    MessageBox.Show("Мигание приостановлено");
}
```

## Как создать анимированный значок в области уведомлений?

Но если вам нужен не мигающий, а анимированный значок в области уведомлений, то тогда вам надо сначала подготовить серию значков для анимации и периодически менять эти значки в свойстве `Icon`. Пример приведен в листинге 6.45.

**Листинг 6.45. Анимация значка**

```
// Массив, содержащий серию значков для создания анимации
private Icon[] aIcons = new Icon[8];

// текущий значок
int curIcon = 0;

private void Form1_Load(object sender, EventArgs e)
{
    // загружаем серию значков
    aIcons[0] = new Icon("moon01.ico");
    aIcons[1] = new Icon("moon02.ico");
    aIcons[2] = new Icon("moon03.ico");
```

```
    aIcons[3] = new Icon("moon04.ico");
    aIcons[4] = new Icon("moon05.ico");
    aIcons[5] = new Icon("moon06.ico");
    aIcons[6] = new Icon("moon07.ico");
    aIcons[7] = new Icon("moon08.ico");
}

private void timer1_Tick(object sender,
{
    notifyIcon1.Icon = aIcons[curIcon];
    curIcon++;
    if (curIcon > 7) curIcon = 0;
}
```

## Свертывание формы вместо закрытия приложения

Как и было обещано в *главе 5*, я расскажу, как реализовать функциональность, имеющуюся у интернет-пейджеров ICQ, Qip, Mail@Agent и других программ. Если вы пользуетесь этими программами, то обратили внимание, что при нажатии на крестик в заголовке программы приложение не закрывается, а просто сворачивается в область уведомлений. Делается это при помощи обработки события `FormClosing`, как показано в листинге 6.46.

**Листинг 6.46. Сворачивание вместо закрытия**

```
private bool m_CloseOK = false;

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    // Пользователь выходит из программы не через контекстное меню
    if (m_CloseOK == false)
    {
        e.Cancel = true;
        this.Hide();
    }
}

private void выходToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
    // выходим из программы по-настоящему
    m_CloseOK = true;
    this.Close();
}
```

Небольшой комментарий к примеру. Сначала нам необходимо создать контекстное меню с командой **Выход** и связать его с элементом `NotifyIcon`. Далее мы создаем флаг `m_CloseOK`, с помощью которого можно определить, как происходит закрытие программы. Если пользователь пытается закрыть форму стандартным способом через крестик в заголовке формы, то мы отменяем его операцию и просто сворачиваем форму в область уведомлений. А если пользователь закрывает программу через соответствующую команду контекстного меню, то тогда действительно закрываем приложение.

> ***Примечание***
>
> Пример использования `NotifyIcon` находится в папке NotifyIconDemo на прилагаемом диске.

# Элемент *ListView*

## Как убрать выделение элемента в *ListView* программно?

Если вы хотите снять выделение с заданного элемента в многостолбцовом списке (`ListView`), то воспользуйтесь методом `Clear`. В листинге 6.47 выделение снимается со всех элементов.

**Листинг 6.47. Снятие выделения с элемента**

```
// Убираем выделение с нужного элемента
this.listView1.SelectedItems.Clear();
```

## Как программно выбрать элемент в *ListView?*

Чтобы выбрать элемент в `ListView`, установите сначала фокус на `ListView` и выберите нужный элемент по его индексу (отчет индекса начинается с 0). Пример показан в листинге 6.48. Если вы забудете установить фокус на `ListView`, то никакого выделения элемента не произойдет.

**Листинг 6.48. Программное выделение элемента в `ListView`**

```
//  Установим фокус
listView1.Focus();
// Выбираем второй элемент
listView1.Items[1].Selected = true;
```

## Как сортировать элемент управления *ListView* по колонкам

Данный совет основывается на статье "How to sort a ListView control by a column in Visual C#" из базы знаний, расположенной по адресу **http://support.microsoft.com/kb/319401/EN-US**.

Когда вы работаете с элементом `ListView`, может потребоваться отсортировать его содержимое, основываясь на значениях в заданной колонке. Пример подобной функциональности имеется в Проводнике, когда вы видите содержимое папки на жестком диске. В режиме Таблица (Details) Проводник показывает информацию о файлах в этой папке. Например, вы видите имя файла, размер файла, тип файла и дату изменения файла. Когда вы щелкаете на заголовке колонки, список сортируется по возрастанию по признаку этой колонки. При повторном щелчке колонка сортируется по убыванию.

В примере, построенном в этой статье, используется класс, унаследованный от интерфейса `IComparer`. Дополнительно в этом примере используется метод `Compare` класса `CaseInsenstiveComparer`. Обратите внимание, что этот метод сравнения не учитывает регистр ("Apple" считается одинаковым с "apple"). Также обратите внимание, что все колонки отсортированы на текстовый манер. Если вы хотите отсортировать по-другому (например, по числам), вы должны заменить следующий код на свой:

```
// Замените этот код на свой
ObjectCompare.Compare(listviewX.SubItems[ColumnToSort].Text,
                      listviewY.SubItems[ColumnToSort].Text);
```

Создайте новый проект Visual C# и добавьте элемент `ListView` на форму. Вставьте следующий код в класс формы:

```
private ListViewColumnSorter lvwColumnSorter;
```

Вставьте код, приведенный в листингах 6.49—6.51, в соответствующие места проекта.

**Листинг 6.49. Код, добавляемый в конструктор формы**

```
// Вставьте код в конструктор формы после вызова
// метода InitializeComponent
// Создаем экземпляр  ListViewColumnSorter и связываем его с ListView
lvwColumnSorter = new ListViewColumnSorter();
this.listView1.ListViewItemSorter = lvwColumnSorter;
```

**Листинг 6.50. Код, добавляемый в обработчик события `Load`**

```
// Вставьте следующий код в обработчик события Load формы

// Используется для создания заголовков
ColumnHeader columnheader;
// Используется для создания элементов в ListView
ListViewItem listviewitem;

// Устанавливаем нужный вид
listView1.View = View.Details;

// Создаем несколько элементов, содержащих имена и фамилии
listviewitem = new ListViewItem("Александр");
listviewitem.SubItems.Add("Суворов");
this.listView1.Items.Add(listviewitem);
listviewitem = new ListViewItem("Наполеон");
listviewitem.SubItems.Add("Бонапарт");
this.listView1.Items.Add(listviewitem);
listviewitem = new ListViewItem("Михаил");
listviewitem.SubItems.Add("Кутузов");
this.listView1.Items.Add(listviewitem);
listviewitem = new ListViewItem("Юлий");
listviewitem.SubItems.Add("Цезарь");
this.listView1.Items.Add(listviewitem);

// Создаем колонки
columnheader = new ColumnHeader();
columnheader.Text = "Имя";
this.listView1.Columns.Add(columnheader);
columnheader = new ColumnHeader();
   columnheader.Text = "Фамилия";
   this.listView1.Columns.Add(columnheader);
```

```
// Проходим через все элементы и устанавливаем размер каждого
// заголовка колонки равным тексту
foreach (ColumnHeader ch in this.listView1.Columns) { ch.Width = -2; }
```

**Листинг 6.51. Код, добавляемый в обработчик события `ColumnClick` элемента `ListView`**

```
// Вставляем следующий код в событие ColumnClick для элемента ListView

// Определяем, является ли колонка, на которой щелкнули, уже
// отсортированной колонкой
if (e.Column == lvwColumnSorter.SortColumn)
{
    // Переворачиваем направление сортировки
    if (lvwColumnSorter.Order == SortOrder.Ascending)
    {
        lvwColumnSorter.Order = SortOrder.Descending;
    }
    else
    {
        lvwColumnSorter.Order = SortOrder.Ascending;
    }
}
else
{
    // Устанавливаем колонку для сортировки
    // По умолчанию сортировка по возрастанию
    lvwColumnSorter.SortColumn = e.Column;
    lvwColumnSorter.Order = SortOrder.Ascending;
}

// Сортируем с новыми настройками
this.listView1.Sort();
```

В меню **Project** щелкните на пункте **Add Class**, чтобы добавить новый класс в проект. Замените весь код по умолчанию в новом классе кодом, приведенном в листинге 6.52.

**Листинг 6.52. Код для нового класса `ListViewColumSorter`**

```
using System.Collections;
using System.Windows.Forms;
```

```
/// <summary>
/// Этот класс реализует интерфейс IComparer.
/// </summary>
public class ListViewColumnSorter : IComparer
{
    /// <summary>
    /// Номер колонки, по которой проводится сортировка
    /// </summary>
    private int ColumnToSort;
    /// <summary>
    /// Порядок, в котором проводится сортировка (например 'Ascending')
    /// </summary>
    private SortOrder OrderOfSort;
    /// <summary>
    /// Объект, проводящий нечувствительную к регистру сортировку
    /// </summary>
    private CaseInsensitiveComparer ObjectCompare;
    /// <summary>
    /// Конструктор. Инициализирует различные элементы
    /// </summary>
    public ListViewColumnSorter()
    {
        // Инициализировать колонку значением 0
        ColumnToSort = 0;
        // Иниацилизировать порядок сортировки значением 'none'
        OrderOfSort = SortOrder.None;
        // Инициализировать объект CaseInsensitiveComparer
        ObjectCompare = new CaseInsensitiveComparer();
    }
    /// <summary>
    /// Этот метод унаследован от интерфейса IComparer. Он сравнивает
    /// два переданных ему объекта, не обращая внимания на регистр
    /// </summary>
    /// <param name="x">Первый объект для сравнения</param>
    /// <param name="y">Второй объект для сравнения</param>
    /// <returns>Результат сравнения. '0' в случае равенства,
    /// отрицательный если 'x' меньше 'y' и положительный
    /// если 'x' больше 'y'</returns>
    public int Compare(object x, object y)
    {
        int compareResult;
        ListViewItem listviewX, listviewY;
```

```
        // Преобразует объекты для сравнения в объекты ListViewItem
        listviewX = (ListViewItem)x;
        listviewY = (ListViewItem)y;
        // Сравнивает 2 значения
        compareResult = ObjectCompare.Compare(
                            listviewX.SubItems[ColumnToSort].Text,
                            listviewY.SubItems[ColumnToSort].Text);
        // Подсчитывает возвращаемый результат, базируясь на результате
        // сравнения
        if (OrderOfSort == SortOrder.Ascending)
        {
            // Выбрана сортировка по возрастанию, возвращаем результат
            // операции сравнения
            return compareResult;
        }
        else if (OrderOfSort == SortOrder.Descending)
        {
            // Выбрана сортировка по убыванию, возвращаем результат
            // операции сравнения со знаком минус
            return (-compareResult);
        }
        else
        {
            // Возвращаем 0, чтобы показать равенство
            return 0;
        }
    }
    /// <summary>
    /// Получает или возвращает номер колонки, к которой применять
    /// сортировку (по умолчанию 0).
    /// </summary>
    public int SortColumn
    {
        set
        {
            ColumnToSort = value;
        }
        get
        {
            return ColumnToSort;
        }
    }
```

```
    /// <summary>
    /// Устанавливает порядок сортировки (например, 'Ascending').
    /// </summary>
    public SortOrder Order
    {
        set
        {
            OrderOfSort = value;
        }
        get
        {
            return OrderOfSort;
        }
    }
}
```

Сохраните и запустите проект. Щелкните на любом заголовке колонки в `ListView`. При этом содержимое `ListView` будет отсортировано по возрастанию. При повторном щелчке данные будут упорядочены по убыванию.

## Изменение цвета подэлементов *ListView* программным путем

Если вы попытаетесь изменить цвет подэлементов в элементе управления `ListView` (свойства `ForeColor` и `BackColor`) программным путем, то можете не увидеть никаких изменений. Вам необходимо сначала установить свойство `UseItemStyleForSubItems` элемента в значение `false`, как сделано в листинге 6.53.

**Листинг 6.53. Изменение цвета элементов в `ListView`**

```
private void button1_Click(object sender, EventArgs e)
{
    listView1.Items.Clear();
    listView1.Items.Add("Алла");
    listView1.Items.Add("София");

    listView1.Items[0].UseItemStyleForSubItems = false;
    listView1.Items[0].SubItems.Add("Пугачева", Color.Pink,
                                    Color.Yellow, Font);
```

```
    listView1.Items[1].UseItemStyleForSubItems = false;
    listView1.Items[1].SubItems.Add("Ротару", Color.Teal,
                                    Color.Violet, Font);
}
```

***Примечание***

Примеры использования элемента `ListView` находятся в папке ListViewDemo на прилагаемом диске.

# Элемент управления *ToolTip*

Элемент управления `ToolTip` в .NET Framework 2.0 был существенно переработан по сравнению с версией .NET Framework 1.1. Теперь у разработчика появилась возможность менять цвет фона и текста во всплывающей подсказке (свойства `BackColor` и `ForeColor`). Также вы можете задать заголовок у подсказки (свойство `ToolTipTitle`), значок (свойство `ToolTipIcon`). Дополнительно вы можете использовать стиль комиксов при помощи свойства `IsBalloon`. Если вы хотите самостоятельно отрисовывать данный элемент, то используйте свойство `OwnerDraw`, установленное в значение `true`.

## Почему пользователь не видит подсказки в стиле Balloon?

Вам хочется использовать красивую подсказку в стиле Balloon, но пользователь почему-то ее не видит? Дело в том, что в Windows XP можно отключить такой вид подсказки через правку реестра. За отключение подсказки отвечает параметр `EnableBalloon` типа DWORD в разделе `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced`. Если значение этого параметра будет равно 0, а вы установите в свойстве `IsBalloon` значение `true`, то пользователь не увидит вашей подсказки. Поэтому рекомендуется проверять указанный параметр реестра, например, способом, предложенным в листинге 6.54.

**Листинг 6.54. Проверка настроек в реестре**

```
if((int)Microsoft.Win32.Registry.GetValue(
@"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced",
```

```
            "EnableBalloonTips", 1) == 0);
        // Не использовать стиль Balloon
        this.Text = "Not use Balloon style";
else
        this.toolTip1.IsBalloon= true;
```

## Многострочная подсказка

Не все догадываются, что всплывающая подсказка может быть многострочной. Раньше, к примеру, программистам на Visual Basic приходилось вызывать системные функции Windows API. Теперь это можно сделать прямо в дизайнере форм. Выберите у элемента управления `toolTip1` свойство `ToolTip` и начинайте вводить текст. Если вам нужно ввести текст со следующей строки, просто нажмите клавишу <Enter>, как вы обычно делаете в любом текстовом редакторе.

Если это можно сделать в дизайнере формы, значит можно сделать и программно. Смотрим исходный код файла Form1.Designer.cs и видим нужную строчку (листинг 6.55).

**Листинг 6.55. Создание многострочной подсказки**

```
this.toolTip1.SetToolTip(this.button1,
                        "Это кнопка\r\nСамая обычная кнопка");
```

***ПРИМЕЧАНИЕ***

Примеры работы с элементом `ToolTip` находятся в папке ToolTipDemo на прилагаемом диске.

# Меню

Меню в .NET Framework представлены такими элементами управления, как `MenuStrip`, `ContextMenuStrip`, `MainMenu`, `ContextMenu`. Причем два последних элемента не отображаются на панели инструментов, так как считаются устаревшими элементами и оставлены для совместимости. А новые элементы имеют очень широкие возможности настройки. Например, теперь не надо писать сложный код, чтобы добавить картинку в пункт меню — достаточно воспользоваться свойством `Image`.

## Фон для меню

Если вы решили использовать элемент `MenuStrip`, то увидите, что его фон имеет синий цвет, схожий с цветовой схемой, используемой в пакете Microsoft Office 2007. Свойство `BackColor` при этом имеет значение `Control`, что может смутить разработчика, который привык, что фон меню обычно совпадает с фоном формы. Исправить эту ситуацию очень просто. Нужно указать другой способ отрисовки элемента при помощи свойства `RenderMode`. Установите в этом свойстве значение `System`, и фон вашего меню будет совпадать с фоном формы. Этот же способ работает и для элемента `ToolStrip`.

## Как добавить контекстное меню элементу управления?

Добавить контекстное меню элементу управления очень просто, но многие программисты даже не догадываются об этой возможности. Для этого сначала нужно добавить на форму элемент `ContextMenuStrip`, а затем просто установить свойство `ContextMenuStrip` у вашего элемента управления. Например, таким образом можно создать контекстное меню для кнопки (хотя и трудно представить ситуацию, когда кнопке необходимо контекстное меню).

## Как определить, какой элемент вызвал контекстное меню?

Если вы сопоставили контекстное меню `ContextMenuStrip` сразу нескольким элементам, то можно определить, какой именно элемент вызвал контекстное меню при помощи свойства `ContextMenu.SourceControl`. Это бывает полезным, если ваше меню должно отличаться для разных элементов. Пример показан в листинге 6.56.

**Листинг 6.56. Определение элемента, вызывающего контекстное меню**

```
private void contextMenuStrip1_Opening(object sender, CancelEventArgs e)
{
    if (contextMenuStrip1.SourceControl == label1)
    {
```

```
        cmenuOpen.Text = "Label";
    }
    else
        cmenuOpen.Text = "Button";
}
```

## Автоматическое закрытие контекстного меню через заданный промежуток времени

Достаточно просто обеспечить автоматическое закрытие контекстного меню через заданный промежуток времени при помощи нажатия на клавишу <Esc> программным способом. Для этого нужно добавить в проект таймер, который и будет посылать команду клавише <Esc>, и написать соответствующий код для события `Opened`. Код приведен в листинге 6.57.

**Листинг 6.57. Автоматическое закрытие контекстного меню**

```
private void timerMenu_Tick(object sender, EventArgs e)
{
    SendKeys.Send("{ESC}");
    timerMenu.Stop();
}

private void contextMenuStrip1_Opened(object sender, EventArgs e)
{
    //set interval to 5 seconds
    timerMenu.Interval = 5000;
    timerMenu.Start();
}
```

Обратите внимание, что в последних двух примерах мы использовали новые события `Opening` и `Opened`, которые появились в .NET Framework 2.0. Для устаревшего элемента `ContexMenu` можно использовать событие `Popup`.

> ***ПРИМЕЧАНИЕ***
>
> Примеры работы с меню находятся в папке MenuDemo на прилагаемом диске.

# Дерево (*TreeView*)

## Как показать подсказку над узлом *TreeView?*

Даже если вы будете использовать элемент `ToolTip` в `TreeView`, вы не сможете показать всплывающую подсказку над конкретным узлом дерева `TreeView`. Здесь требуется обходной маневр, описанный в Базе Знаний Microsoft (**http://support.microsoft.com/kb/322634**). Предлагаю вашему вниманию чуточку измененный вариант примера из этой статьи (листинг 6.58). Результат показан на рис. 6.9.

**Листинг 6.58. Показ подсказки над узлом `TreeView`**

```
private void Form1_Load(object sender, EventArgs e)
{
    // Create a root node.
    TreeNode rootNode = treeView1.Nodes.Add("Коты");

    TreeNode childNode = rootNode.Nodes.Add("Барсик");
    childNode.Tag = "Барсик - большой и умный кот";

    childNode = rootNode.Nodes.Add("Рыжик");
    childNode.Tag = "Рыжик - очень любопытный кот";

    childNode = rootNode.Nodes.Add("Мурзик");
    childNode.Tag = "Мурзик - ленивый кот";

    childNode = rootNode.Nodes.Add("Пушок");
    childNode.Tag = "Пушок - белый и пушистый кот";

    // Раскрываем все узлы дерева
    rootNode.ExpandAll();
}

private void treeView1_MouseMove(object sender, MouseEventArgs e)
{
    // Получим узел в текущей позиции мыши
    TreeNode theNode = this.treeView1.GetNodeAt(e.X, e.Y);

    // Установим ToolTip, только если мышь задержалась на узле
    if ((theNode != null))
```

```
    {
        // Проверяем, что свойство tag не пустое
        if (theNode.Tag != null)
        {
            // Меняем ToolTip, если мышь переместилась на другой узел
            if (theNode.Tag.ToString() !=
                    this.toolTip1.GetToolTip(this.treeView1))
            {
                this.toolTip1.SetToolTip(this.treeView1,
                                         theNode.Tag.ToString());
            }
        }
        else
        {
            this.toolTip1.SetToolTip(this.treeView1, "");
        }
    }
    else     // Если указатель не над узлом, то очистим подсказку
    {
        this.toolTip1.SetToolTip(this.treeView1, "");
    }
}
```

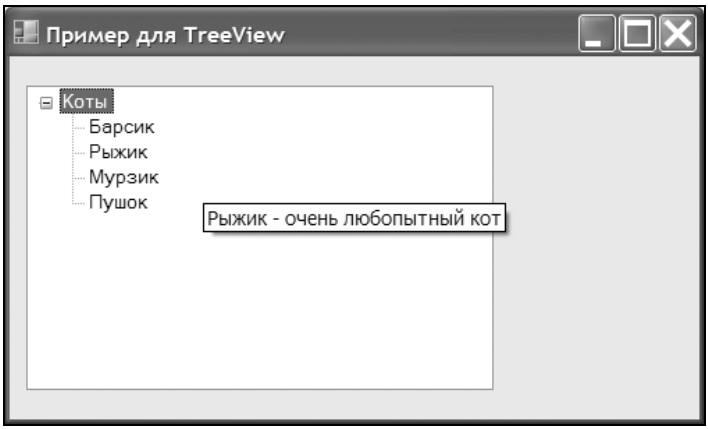



**Рис. 6.9.** Показ подсказки над узлом TreeView

### *Примечание*

Пример находится в папке TreeViewDemo на прилагаемом компакт-диске.

# Вкладки (*TabControl*)

## Программное переключение на другую вкладку

Чтобы программно переключаться на разные вкладки элемента управления `TabControl`, можно использовать любой из двух способов — или использование свойства `SelectedTab`, или использование свойства `SelectedIndex`. Оба варианта приведены в листинге 6.59.

**Листинг 6.59. Программное переключение на другую вкладку**

```
private void button1_Click(object sender, EventArgs e)
{
    // Переключаемся на другую вкладку при помощи SelectedTab
    this.tabControl1.SelectedTab = this.tabPage2;
}

private void button2_Click(object sender, EventArgs e)
{
    // Переключаемся на другую вкладку при помощи SelectedIndex
    this.tabControl1.SelectedIndex = 0;
}
```

## Установка фокуса на элементе управления на вкладке во время загрузки формы

Если вам нужно во время загрузки формы установить фокус на элементе управления, который находится на вкладке, то используйте для этой цели событие `Activated`, а не `Load` (листинг 6.60).

**Листинг 6.60. Установка фокуса на элементе управления**

```
private void Form1_Load(object sender, EventArgs e)
{
    // не работает
    this.button1.Focus();
}
```

```
private void Form1_Activated(object sender, EventArgs e)
{
    // работает
    this.button1.Focus();
}
```

## Как вывести ярлычки внизу вкладки *TabControl?*

Чтобы ярлычки (`tabPage`) на экране были в нижней части вкладки, то установите свойство `Alignment` в значение `Bottom`:

```
tabControl1.Alignment = TabAlignment.Bottom;
```

Также данное свойство может принимать значения `Left` и `Right`, которые позволяют расположить ярлычки слева или справа.

## Добавление новой вкладки

Если вам нужно программно добавить новую вкладку, то используйте метод `Add` (листинг 6.61).

**Листинг 6.61. Добавление новой вкладки**

```
// Добавляем новую вкладку
tabControl1.Controls.Add(new TabPage("Новая вкладка"));
```

Новая вкладка всегда добавляется как последняя вкладка. Если вы нажмете на кнопку, в обработчике события `Click` которой указан этот код, еще раз, то будет создана еще одна вкладка. И так далее.

## Удаление вкладки

Решить противоположную задачу — удалить вкладку — можно с помощью метода `Remove` (листинг 6.62).

**Листинг 6.62. Удаление вкладки**

```
// Удалить выбранную вкладку
tabControl1.Controls.Remove(tabControl1.SelectedTab);
```

## Как вставить вкладку в определенную позицию?

Резонно возникает вопрос: а как вставить вкладку не в конец, а в определенную позицию? Напрямую, встроенными средствами эту задачу не решить. Пойдем обходным путем и создадим собственную процедуру (листинг 6.63), позволяющую вставлять ярлычок вкладки в любую позицию.

**Листинг 6.63. Вставка вкладки в определенную позицию**

```
/// <summary>
/// Вставляет вкладку в заданную позицию
/// </summary>
/// <param name="tabNumber">Номер вкладки</param>
/// <param name="tabControl">Элемент TabControl</param>
private void InsertTab(int tabNumber, ref TabControl tabControl)
{
    int counter = tabControl.Controls.Count;
    if (tabNumber < 0 || tabNumber > counter)
    {
        tabControl.Controls.Add(new TabPage("Вкладка"));
        return;
    }

    int target = tabControl.SelectedIndex;

    // сохраняем существующие ярлычки и очищаем элементы
    Control[] c = new Control[counter];
    tabControl.Controls.CopyTo(c, 0);
    tabControl.Controls.Clear();

    // Добавляем ярлычки до вставляемого элемента
    for (int i = 0; i < target; ++i)
        tabControl.Controls.Add(c[i]);
    // Вставляем свой ярлычок
    tabControl.Controls.Add(new TabPage("Вставленная вкладка"));

    // Добавляем ярлычки после вставляемого элемента
    for (int i = target; i < counter; ++i)
        tabControl.Controls.Add(c[i]);
```

```
    // Выбираем вставленную вкладку
    tabControl.SelectedIndex = target;
}

private void button5_Click(object sender, EventArgs e)
{
    // Добавляем ярлычок во вторую позицию
    InsertTab(2, ref tabControl1);
}
```

***ПРИМЕЧАНИЕ***

Примеры работы с элементом `TabControl` находятся в папке TabControl на прилагаемом диске.

# Элемент *PerformanceCounter*

Элемент `PerformanceCounter` является удобным средством для отслеживания производительности ресурсов системы или получения данных о производительности приложения. Возможности элемента `PerformanceCounter` очень обширны, поэтому остановимся только на одном примере.

## Как создать счетчик производительности процессора?

Создать приложение, которое будет отслеживать загрузку процессора, можно буквально за пару минут, что является еще одним свидетельством могущества платформы .NET Framework. Разместите на форме элементы `ProgressBar`, `Label`, `Timer` и `PerformanceCounter`. Далее установим необходимые свойства у этих элементов. Элемент `Label` оставляем без изменений. Все необходимые значения у него будут устанавливаться программным путем. У индикатора прогресса установим значение свойства `Step` равным `1`. Для таймера установим значение свойства `Interval` равным `1000` и сделаем его доступным при запуске приложение (`Enabled = True`). Первоначальные приготовления завершены. Теперь перейдем к набору счетчиков `PerformanceCounter`. Вся его настройка может быть осуществлена через окно свойств дизайнера форм (рис. 6.10).

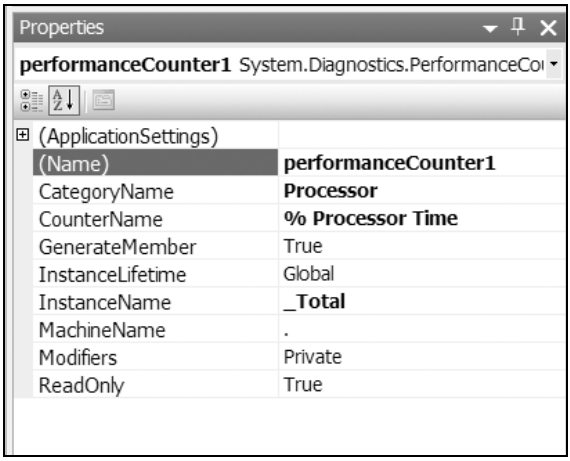



**Рис. 6.10.** Окно свойств для `PerfomanceCounter`

Выберите свойство `CategoryName`, далее из списка выберите пункт `Processor`. Кстати, обратите внимание, какое большое количество возможных категорий для счетчиков можно выбрать из данного списка. Теперь переходим к свойству `CounterName`. Так как в предыдущем шаге мы выбрали уже конкретную категорию, то здесь нам будет предложен для выбора список счетчиков производительности, относящихся к процессорам. Давайте выберем значение `% Processor Time` и установим в свойстве `InstanceName` значение `_Total`. Если у вас в системе установлено несколько процессоров, то в этом свойстве можете указать, какой процессор вас интересует. Осталось только написать пару строчек кода (листинг 6.64), и приложение готово!

**Листинг 6.64. Проверка загрузки процессора**

```
private void timer1_Tick(object sender, EventArgs e)
{
    progressBar1.Value = (int)(performanceCounter1.NextValue());
    label1.Text = "Загрузка процессора: " +
                  progressBar1.Value.ToString() + "%";
}
```

Запустив программу, вы увидите текущую нагрузку процессора в реальном времени на своей машине (рис. 6.11). Обратите внимание, что у элемента `PerformanceCounter` имеется свойство `MachineName`, которое позволяет снимать показания не только с локальной машины, но и с удаленного компьюте-

ра. Таким образом, вы можете написать специальную программу для системных администраторов.

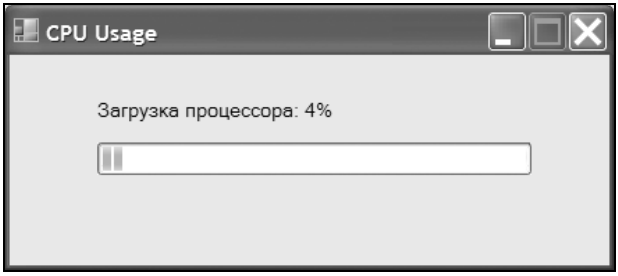



**Рис. 6.11.** Приложение, показывающее работу процессора

***ПРИМЕЧАНИЕ***

Пример находится в папке PerformanceCounter на прилагаемом диске.

# *StatusBar* и *StatusStrip*

На смену устаревшему элементу управления `StatusBar` пришел новый элемент `StatusStrip`, который обладает новыми возможностями, гораздо удобнее в настройках, и обладает большей функциональностью. Но, возможно, вам по ряду причин по-прежнему приходится использовать `StatusBar`.

## Как изменить шрифт и фон для *StatusBar*

Если вы вынуждены использовать в своих проектах старый элемент `StatusBar` и хотите поменять шрифт или фон для этого элемента, то обратитесь к статье Базы Знаний Microsoft "How to change the color and the font of the StatusBarPanel object by using Visual C#", которая находится по адресу **http://support.microsoft.com/kb/319311/**. В этой статье рассказывается, что для элемента `StatusBar` нужно установить свойство `Style` в значение `OwnerDraw` и самостоятельно управлять шрифтом и цветом в программе. Для `StatusStrip` аналогичная задача решается элементарно через свойства элемента.

# Элементы *FlowLayoutPanel* и *TableLayoutPanel*

Два новых элемента управления, которые появились в .NET Framework 2.0, являются контейнерами, позволяющими очень быстро разместить множество элементов управления в организованном порядке. Чтобы получить краткое представление о том, как они работают, сделайте следующее. Поместите на форме элемент `FlowLayoutPanel`, растяните его по ширине формы, и щелкните несколько раз подряд на элементе `Button` на панели инструментов, чтобы добавить кнопки на форму. Вы увидите, что кнопки автоматически выстраиваются друг за другом. В сложном проекте, где требуется разместить множество элементов управления, контейнер `FlowLayoutPanel` позволит сэкономить время на подгонку элементов друг к другу. Более детальное рассмотрение новых элементов управления я оставляю вам для самостоятельного изучения.

# Элемент *DataGrid*

Честно говоря, я в своей практике не часто пользовался этим элементом, поэтому я лишь приведу заголовки советов о данном элементе, найденные в сборнике Вопросов и ответов на сайте **http://www.gotdonet.ru**. Если вас заинтересуют названия советов, то отправляйтесь за подробностями на этот сайт.

- Как прокрутить `DataGrid` к строке с определенным индексом?
- Как отобразить в элементах управления отдельной формы текущую строку `DataGrid`?
- Как перехватывать сообщения мыши для определенной колонки `DataGrid`?
- Как отследить нажатия кнопок в определенной колонке `DataGrid`?
- Как изменить текст ячейки `DataGrid`?
- Как, зная координаты ячейки `DataGrid`, считать или записать ее значение?
- Как запретить пользователю изменять ширину столбцов и высоту строк `DataGrid`?

- Как сделать, чтобы в ячейках `DataGrid` значения отображались в нужном формате?
- Как в `DataGrid` сделать текущей строку, по значению любого ее столбца в независимости от сортировки (источником данных для `DataGrid` является `DataTable`)?
- Как в ячейку `DataGrid` поместить `ComboBox` или `Button`?
- Какое событие возникает при перемещении по строкам `DataGrid`?
- Как привязать стили к `DataGrid`, если источником данных (`DataSource`) является не таблица, а `ArrayList` или массив?
- Как вывести подтверждающее диалоговое окно при попытке пользователя удалить строку путем нажатия кнопки <Del> в `DataGrid`?
- Как для `DataGrid` получить `CurrencyManager`?
- Как отследить нажатие клавиши на `DataGrid`? Для элемента управления `DataGrid` был создан обработчик события `KeyDown` и `KeyUp`, однако этот обработчик не получает управление во время редактирования данных.
- Как в `DataGrid` убрать последнюю строку со звездочкой, запретив тем самым добавление новых записей?
- Как в `DataGrid` программно задать ширину столбца/скрыть его/изменить текст его заголовка/переместить?
- Как получить `DataView`, используемый `DataGrid`?
- У `DataGrid` в качестве источника данных установлен `DataSet`/`DataTable`/`DataView`, как определить текущую строку?

# Элемент *DataGridView*

Новый элемент управления `DataGridView`, появившийся в .NET Framework 2.0, достоин описания в отдельной книге. Судите сами, данный элемент содержит 153 свойства, 87 методов и 187 событий! Советую обратиться к статье "DataGridView. Новый контрол в составе Framework 2.0", которая находится по адресу **http://www.rsdn.ru/article/dotnet/DataGridView20.xml**, для более глубокого знакомства с этим сложным элементом.

# Создание собственных элементов управления

Кроме использования готовых элементов управления, входящих в поставку Visual Studio, программистам часто приходится разрабатывать свои собственные элементы управления. Далее мы расскажем о некоторых приемах программирования, применяемых при создании таких элементов.

## Как скрыть свойство или метод от IntelliSense в редакторе кода?

Для этого предназначен атрибут `EditorBrowsableAttribute`. Его применение проиллюстрировано в листинге 6.65.

**Листинг 6.65. Скрытие свойства или метода от IntelliSense**

```
//...
string firstNameValue;
[EditorBrowsableAttribute(EditorBrowsableState.Never)]
public string FirstName
{
     get
 {
      return firstNameValue;
 }
 set
 {
      firstNameValue = value;
 }
}
//...
```

Нужно отметить, что на самом деле такое свойство или метод никуда не исчезает, и если его все-таки набрать в редакторе, то с ним можно успешно продолжать работать.

# Как скрыть свойства и события из редактора свойств *PropertyGrid* при создании собственного элемента управления?

По умолчанию при разработке собственного элемента все открытые свойства и события отображаются в редакторе свойств **PropertyGrid**. Если вы не хотите отображать некоторые из открытых свойств или событий, то прикрепите к соответствующему члену класса атрибут `BrowsableAttribute`, передав в конструктор значение `false` (листинг 6.66).

**Листинг 6.66. Скрытие свойств и событий из редактора свойств PropertyGrid**

```
[BrowsableAttribute(false)]
public string FirstName
{
   get
   {
     return firstNameValue;
   }
   set
   {
     firstNameValue = value;
   }
}
[BrowsableAttribute(false)]
public event EventHandler FirstNameChanged;
```

Если же нужно скрыть свойства или события, унаследованные от базового класса, создайте класс, наследующий `System.Windows.Forms.Design.ControlDesigner`, и, переопределяя методы `PostFilterProperties` и `PostFilterEvents`, удаляйте в них соответствующие свойства и события. Полученный класс укажите как класс-дизайнер для класса элемента управления. Сделать это можно при помощи атрибута `System.ComponentModel.DesignerAttribute`. Пример в листинге 6.67 показывает, как из элемента управления `UserControlDemo` "удаляются" свойства `BackColor`, `BackgroundImage`, а также события `Click` и `DoubleClick`.

**Листинг 6.67. Скрытие некоторых свойств и событий элемента управления**

```
[Designer(typeof(ControlDesignerEx))]
public class UserControlDemo: UserControl
{
   //...
}
class ControlDesignerEx: ControlDesigner
{
  protected override void PostFilterEvents(
                              System.Collections.IDictionary events)
  {
    events.Remove("Click");
    events.Remove("DoubleClick");
    base.PostFilterEvents (events);
  }
  protected override void PostFilterProperties(
                              System.Collections.IDictionary properties)
  {
    properties.Remove("BackColor");
    properties.Remove("BackgroundImage");
    base.PostFilterProperties (properties);
  }
}
```

## Как запретить изменять размер элемента управления во время разработки?

Нужно создать свой класс `Designer` (унаследованный от `ControlDesigner`) и переопределить в нем свойство `SelectionRules`, как показано в листинге 6.68.

**Листинг 6.68. Запрет на изменение размеров элемента управления**

```
class MyControlDesigner: ControlDesigner
{
    public override SelectionRules SelectionRules
   {
       get
      {
          return base.SelectionRules & ~SelectionRules.AllSizeable;
```

```
            //Если нужно запретить изменять размер по вертикали,
            //то замените строчку выше на такую:
            //return base.SelectionRules & ~(SelectionRules.BottomSizeable |
                                              SelectionRules.TopSizeable);
        }
    }
}
```

Затем использовать этот класс можно при помощи атрибута `DesignerAttribute` (листинг 6.69).

**Листинг 6.69. Применение созданного класса**

```
[Designer(typeof(MyControlDesigner))]
public class MyUserControl: UserControl
{
 //...
}
```

## Как во время разработки позволить выбирать значение свойства из нескольких предопределенных в поле со списком?

Свойство должно иметь тип перечисления. В примере (листинг 6.70) создается свойство, имеющее тип перечисления, которое описывает наиболее популярные .NET-совместимые языки.

**Листинг 6.70. Выбор значений из поля со списком**

```
public class MyUserControl: UserControl
{
 //Определяем перечисление
 public enum NetLanguage{CSharp, VisualBasic, ManagedCPP};
 NetLanguage myProperty;
 public NetLanguage MyProperty
 {
  get
  {
   return myProperty;
  }
```

```
  set
  {
   myProperty = value;
  }
 }
}
```

## Как добиться того, чтобы свойство моего элемента управления было видно в разделе *DataBindings* окна свойств?

Для этого нужно присоединить к свойству атрибут `Bindable`, как сделано в листинге 6.71.

**Листинг 6.71. Показа свойства в разделе `DataBindings`**

```
public class MyUserControl: UserControl
{
 int myProperty;

 [Bindable(true)]
 public int MyProperty
 {
  get
  {
   return myProperty;
  }
  set
  {
   myProperty = value;
  }
 }
}
```

## Как сделать свой элемент управления, выступающий в роли контейнера для других элементов управления во время разработки?

Необходимо к классу элемента управления присоединить атрибут `DesignerAttribute`, как это сделано в листинге 6.72.

**Листинг 6.72. Использование элемента управления в роли контейнера**

```
[Designer(typeof(ParentControlDesigner))]
public class MyUserControl: UserControl
{
 //...
}
```

## Как присвоить свой значок для собственного элемента управления в панели инструментов?

Чтобы ваш собственный элемент выглядел профессионально и не затерялся среди множества других элементов управления на панели инструментов, нужно создать собственный значок и присвоить его разработанному элементу. Вот несколько правил, которых нужно придерживаться при создании собственного значка. Размеры растровой картинки или значка должны быть размером 16×16 с 16 цветами. Левый нижний пиксел используется для определения прозрачного цвета.

Для этой цели служит атрибут `System.Drawing.ToolboxBitmapAttribute`, который нужно прикрепить к классу создаваемого элемента управления (листинг 6.73). Вот какие операции нужно выполнить для добавления своего значка:

1. Добавьте значок в проект как ресурс (embedded resource). Для этого в окне **Solution Explorer** щелкните на файле со значком правой кнопкой и в контекстном меню выберите команды **Properties | Build Actions | Embedded Resource**.
2. Задайте новому классу атрибут **ToolboxBitmap**.
3. Удостоверьтесь, что свойство проекта **Default Namespace** соответствует пространству имен искомого класса.
4. Перекомпилируйте проект.

**Листинг 6.73. Добавление собственного значка для своего элемента управления**

```
namespace MyClassLibrary
{
    [ToolboxBitmap(typeof(MyControl), "One.ICO")]
```

```
    public class MyControl: Control
    {
        //...
    }
    //...
}
```

Обратите внимание на три вещи. Первый параметр этого варианта конструктора класса `ToolboxBitmapAttribute` принимает объект `Type`, который нужен только для того, чтобы определить сборку, в которой находится требуемый ресурс. Следовательно, это может быть любой тип, определенный в сборке проекта. Второй параметр конструктора чувствителен к регистру. Если файл значка находится в каталоге, отличном от каталога проекта (например, во вложенной папке) и этот файл был добавлен в проект через меню **Open | Link File** (то есть без копирования в каталог проекта), то при указании имени изображения во втором параметре конструктора нужно добавить к названию файла имя папки через точку (например, `"Images.One.ICO"`).

# Создание собственного элемента управления *SmoothProgressBar*

В заключение мы приведем пример создания собственного элемента управления `SmoothProgressBar`, который будет являться реализацией индикатора прогресса с собственным стилем, отличающимся от стандартного индикатора прогресса `ProgressBar`, который входит в состав .NET Framework. Данный пример основан на статье "How to create a smooth progress bar in Visual C# 2005 or in Visual C# .NET" из Базы Знаний Microsoft, которую вы можете найти по адресу **http://support.microsoft.com/kb/323116/**. В ранних версиях элемента управления `ProgressBar` существовало два вида индикатора прогресса. Если у вас установлен пакет Visual Basic 6.0, то вы можете увидеть эти два различных стиля работы элементы. Запустите Visual Basic 6.0, выберите в меню **Project | Components…** и в диалоговом окне **Components** выберите пункт **Microsoft Windows Common Controls 6.0 (SP6)**, который добавит в проект группу элементов ActiveX, в число которых входит и ProgressBar (рис. 6.12).

Далее на панели инструментов найдите элемент `ProgressBar` и добавьте его на форму. В окне свойств у индикатора прогресса вы увидите свойство `Scrolling`, состоящее из двух значений — `ccScrollingStandart` (Стандартный) и `ccScrollingSmooth` (Плавный).

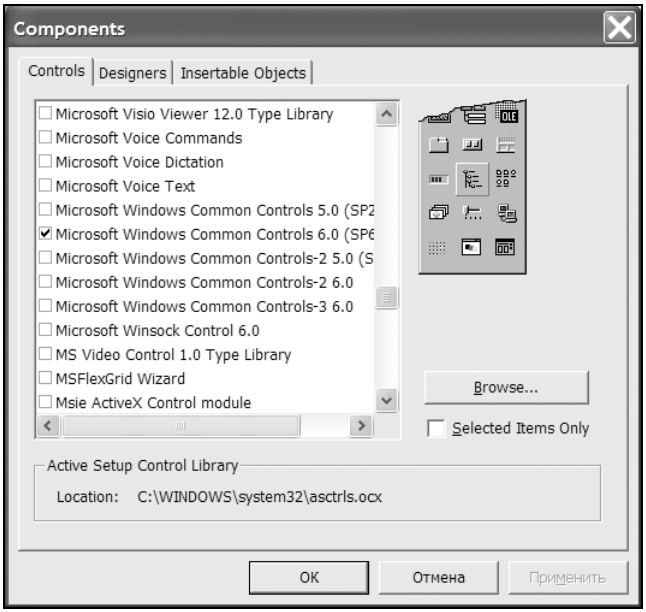



**Рис. 6.12.** Диалоговое окно `Components` в Visual Basic 6.0

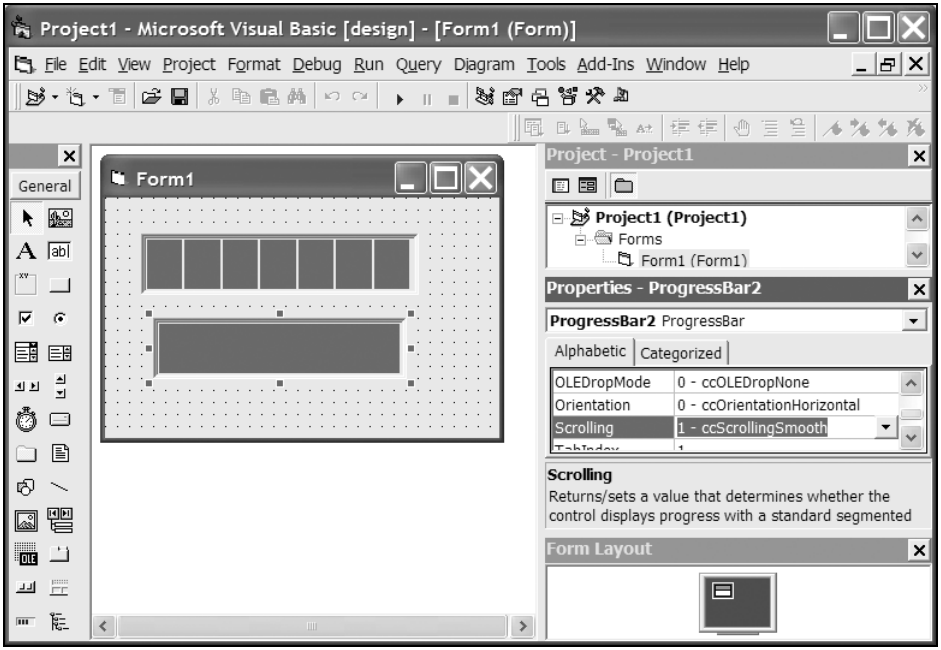



**Рис. 6.13.** Два вида индикатора прогресса в Visual Basic

Плавный стиль заполняет индикатор прогресса сплошной цветной полосой, а стандартный стиль имеет вид маленьких блоков, состоящих из прямоугольников (рис. 6.13).

## Создание элемента *SmoothProgressBar*

Но вернемся к Visual C# 2008. Платформа .NET Framework поддерживает только стандартный стиль для индикатора прогресса. Наша цель — разработать свой элемент управления, который будет поддерживать следующие свойства:

- `Minimum` — свойство, позволяющее получать или устанавливать минимальное из допустимых значений индикатора прогресса. По умолчанию данное свойство будет установлено в 0. Использовать отрицательные значения в этом свойстве недопустимо;
- `Maximum` — свойство, позволяющее получать или устанавливать максимальное из допустимых значений индикатора прогресса. По умолчанию значение данного свойства равно 100;
- `Value` — это свойство получает или устанавливает текущее значение индикатора прогресса. Значение этого свойства не должно выходить за диапазон, ограниченный свойствами `Minimum` и `Maximum`;
- `ProgressBarColor` — это свойство получает или устанавливает цвет полоски индикатора прогресса.

Опишем процесс создания собственного элемента управления `SmoothProgressBar` по шагам.

1. Запустите Microsoft Visual Studio 2008.
2. В меню **File** выберите команды **New** | **Project**.
3. В диалоговом окне **New Project** выберите раздел **Visual C#** | **Windows** и в списке шаблонов выберите **Windows Control Library**.
4. В поле **Name** введите новое имя `SmoothProgressBar` и нажмите кнопку **OK**.
5. В окне **Project Explorer** замените имя по умолчанию для модуля класса с UserControl1.cs на SmoothProgressBar.cs.

Первая часть приготовлений к созданию индикатора прогресса закончена. Теперь приступим к написанию кода для разрабатываемого элемента. Откройте редактор кода для файла SmoothProgressBar.cs и вставьте код, приведенный в листинге 6.74.

**Листинг 6.74. Установка необходимых свойств у собственного индикатора прогресса**

```
int min = 0;        // Minimum value for progress range
int max = 100;      // Maximum value for progress range
int val = 0;        // Current progress
Color BarColor = Color.Blue;        // Color of progress meter

protected override void OnResize(EventArgs e)
{
    // Invalidate the control to get a repaint.
    this.Invalidate();
}

protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;
    SolidBrush brush = new SolidBrush(BarColor);
    float percent = (float)(val - min) / (float)(max - min);
    Rectangle rect = this.ClientRectangle;

    // Вычислим район, в котором рисуем индикатор.
    rect.Width = (int)((float)rect.Width * percent);

    // Нарисуем индикатор.
    g.FillRectangle(brush, rect);

    // Нарисуем 3D-рамку вокруг индикатора.
    Draw3DBorder(g);

    // Освободим ресурсы.
    brush.Dispose();
    g.Dispose();
}

public int Minimum
{
    get
    {
        return min;
    }
```

```
    set
    {
        // Запретим отрицательные значения.
        if (value < 0)
        {
            min = 0;
        }

        // Убедимся, что минимум никогда не устанавливается большим,
        // чем максимум.
        if (value > max)
        {
            min = value;
            min = value;
        }

        // Проверим, что значение все еще внутри допустимого диапозона.
        if (val < min)
        {
            val = min;
        }

        // Запросим перерисовку элемента.
        this.Invalidate();
    }
}

public int Maximum
{
    get
    {
        return max;
    }

    set
    {
        // Убедимся, что максимум не устанавливается меньше минимума.
        if (value < min)
        {
            min = value;
        }
```

```
        max = value;

        // Убедимся, что значение остается в допустимом интервале.
        if (val > max)
        {
            val = max;
        }

        // Запросим перерисовку элемента.
        this.Invalidate();
    }
}

public int Value
{
    get
    {
        return val;
    }

    set
    {
        int oldValue = val;

        // Убедимся, что значение в допустимом интервале.
        if (value < min)
        {
            val = min;
        }
        else if (value > max)
        {
            val = max;
        }
        else
        {
            val = value;
        }

        // Запросим перерисовку только измененной области.
        float percent;

        Rectangle newValueRect = this.ClientRectangle;
        Rectangle oldValueRect = this.ClientRectangle;
```

```
        // Используем новое значение, чтобы получить новый прямоугольник.
        percent = (float)(val - min) / (float)(max - min);
        newValueRect.Width = (int)((float)newValueRect.Width * percent);

        // Используем старое значение, чтобы получить
        // старый прямоугольник.
        percent = (float)(oldValue - min) / (float)(max - min);
        oldValueRect.Width = (int)((float)oldValueRect.Width * percent);

        Rectangle updateRect = new Rectangle();

        // Найдем только кусок, который нужно обновить.
        if (newValueRect.Width > oldValueRect.Width)
        {
            updateRect.X = oldValueRect.Size.Width;
            updateRect.Width = newValueRect.Width - oldValueRect.Width;
        }
        else
        {
            updateRect.X = newValueRect.Size.Width;
            updateRect.Width = oldValueRect.Width - newValueRect.Width;
        }

        updateRect.Height = this.Height;

        // Обновим только пересечение.
        this.Invalidate(updateRect);
    }
}

public Color ProgressBarColor
{
    get
    {
        return BarColor;
    }

    set
    {
        BarColor = value;
```

```
        // Запросим перерисовку.
        this.Invalidate();
    }
}

private void Draw3DBorder(Graphics g)
{
    int PenWidth = (int)Pens.White.Width;

    g.DrawLine(Pens.DarkGray,
        new Point(this.ClientRectangle.Left, this.ClientRectangle.Top),
        new Point(this.ClientRectangle.Width - PenWidth,
                  this.ClientRectangle.Top));
    g.DrawLine(Pens.DarkGray,
        new Point(this.ClientRectangle.Left, this.ClientRectangle.Top),
        new Point(this.ClientRectangle.Left,
                  this.ClientRectangle.Height - PenWidth));
    g.DrawLine(Pens.White,
        new Point(this.ClientRectangle.Left,
                  this.ClientRectangle.Height - PenWidth),
        new Point(this.ClientRectangle.Width - PenWidth,
                  this.ClientRectangle.Height - PenWidth));
    g.DrawLine(Pens.White,
        new Point(this.ClientRectangle.Width - PenWidth,
                  this.ClientRectangle.Top),
        new Point(this.ClientRectangle.Width - PenWidth,
                  this.ClientRectangle.Height - PenWidth));
}
```

В меню **Build** выберите команду **Build Solution**, чтобы скомпилировать проект. На этом заканчивается первая подготовительная часть работы. Далее нам нужно протестировать работу созданного индикатора прогресса.

## Создание клиентской программы для тестирования

Теперь нам нужно протестировать созданный элемент. В меню **File** выберите команду **Add | New Project** и в диалоговом окне **Add New Project** выберите раздел **Visual C# | Windows** в списке **Project Types**. Выберите шаблон **Win-**

**dows Application** и щелкните на кнопке **OK**. Далее опишем процесс добавления двух экземпляров элемента управления `SmoothProgressBar` на форму.

1. В меню **Tools** выберите **Choose Toolbox Items**.
2. Выберите вкладку **.NET Framework Components**.
3. Щелкните на кнопке **Browse** и найдите файл SmoothProgressBar.dll, созданный нами ранее. Нажмите кнопку **OK**.
4. Обратите внимание, что элемент `SmoothProgressBar` был добавлен на панель инструментов.
5. Перетащите два экземпляра элемента `SmoothProgressBar` на форму.

Также добавьте в проект элемент управления `Timer`. Добавьте код, представленный в листинге 6.75, в обработчик события `Tick` элемента `Timer`.

**Листинг 6.75. Код для таймера**

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (this.smoothProgressBar1.Value > 0)
    {
        this.smoothProgressBar1.Value--;
        this.smoothProgressBar2.Value++;
    }
    else
    {
        this.timer1.Enabled = false;
    }
}
```

Теперь добавьте на форму кнопку и напишите код обработчика нажатия (листинг 6.76).

**Листинг 6.76. Код для кнопки**

```
private void button1_Click(object sender, EventArgs e)
{
    this.smoothProgressBar1.Value = 100;
    this.smoothProgressBar2.Value = 0;

    this.timer1.Interval = 1;
    this.timer1.Enabled = true;
}
```

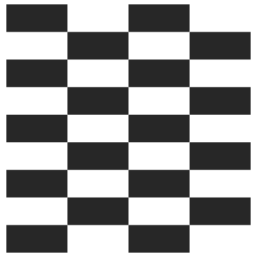
Щелкните на названии проекта **WindowsApplication1** правой кнопкой и в контекстном меню выберите команду **Set as StartUp Project**. В меню **Debug** выберите команду **Start Debugging**, чтобы запустить созданное приложение и проверить работу созданного элемента. В запущенной программе щелкните на кнопке. Вы увидите работу созданных элементов управления в действии. У одного элемента полоска индикатора будет бежать слева направо, а у другого — в обратном направлении. Вы можете настроить различные свойства элемента под себя. Например, через свойство `ProgressBarColor` можно установить цвет индикатора.

> ***Примечание***
>
> Файлы, относящиеся к индикатору прогресса `SmoothProgressBar`, находятся в папке SmoothProgressBar на прилагаемом диске.

# Заключение

В этой главе мы рассмотрели работу с некоторыми элементами интерфейса и заострили внимание на нестандартных действиях, которые можно с ними выполнять. Многие стандартные действия мы не рассматривали, поскольку они выполняются таким же образом, как и в других средах разработки. Также мы опустили описание многих реже встречающихся элементов интерфейса, присутствующих в .NET Framework. Для ознакомления с их работой обратитесь к документации и примерам, предоставляемым фирмой Microsoft и сторонними разработчиками.

# Глава 7



# Графика

Работа с графикой — одна из самых интересных и сложных тем в программировании. В этой главе мы рассмотрим некоторые трюки, связанные с выводом графической информации.

## Преобразование цвета в HTML-формат

Класс `ColorTranslator` пространства имен `System.Drawing` позволяет преобразовать цвет `Color` в строку в формате HTML и обратно. Его применение проиллюстрировано в листингах 7.1 и 7.2.

**Листинг 7.1. Преобразование строки в значение цвета**

```
string htmlColor = "Blue";
// Преобразуем цвет htmlColor в структуру GDI+ Color
Color myColor = ColorTranslator.FromHtml(htmlColor);
butColorToString.BackColor = myColor;

// или так
Color myColor2 = ColorTranslator.FromHtml("#AFFFFF");
butColorToString.BackColor = myColor2;
```

Обратная задача преобразования структуры `Color` в строку также не представляет трудностей.

**Листинг 7.2. Преобразование структуры Color в строку**

```
// Преобразуем структуру Color в строку
Color myColor = Color.Blue;
string htmlColor = ColorTranslator.ToHtml(myColor);
```

```
// Выводим окно сообщения с полученным значением
MessageBox.Show(htmlColor);
```

# Как преобразовать цвет в целое число?

Для преобразования цвета в целое число и обратно используйте методы `FromArgb` и `ToArgb` (листинг 7.3).

**Листинг 7.3. Преобразование цвета в числовое значение**

```
int iBlueColor = Color.Blue.ToArgb();
Color myColor = Color.FromArgb(0x7800FF00);
```

# Как получить доступ к определенному пикселу изображения?

Чтобы получить или установить цвет заданного пиксела изображения, нужно воспользоваться методами `GetPixel` и `SetPixel`, в которых можно указать координаты пиксела и желаемый цвет (листинг 7.4).

**Листинг 7.4. Установка цвета в заданной точке**

```
private void butSetPixel_Click(object sender, EventArgs e)
{
    Bitmap bm = new Bitmap(pictureBox1.Image);

    pictureBox1.Image = bm;
    for (int i = 0; i < 400; i++)
    {
        bm.SetPixel(i, i, Color.Red);
    }
    pictureBox1.Update();
}
```

# Как нарисовать точку?

В .NET Framework нет графического метода, рисующего точку. Но мы можем нарисовать очень маленький прямоугольник, который будет играть роль точки (листинг 7.5).

**Листинг 7.5. Рисование точки**

```
Graphics g = CreateGraphics();
// Рисуем ряд красных точек
for (int i = 0; i < 100; i+=5)
    g.FillRectangle(new SolidBrush(Color.Red), i, 10, 1, 1);
g.Dispose();
```

***Примечание***

Примеры работы с цветом находятся в папке AboutColor на прилагаемом диске.

# Как получить цвет любой выбранной точки экрана?

В этом примере мы сможем получить цвет любой точки на экране с помощью указателя мыши. Для этого добавьте на форму метку `Label` с именем `label1`, которая будет принимать цвет выбранной точки. Основные действия будут производиться при обработке событий метки `MouseDown` и `MouseUp`. В `label1_MouseDown` мы будем получать изображение текущего экрана, а в `label1_MouseUp` получим цвет, связанный с текущей позицией пиксела, на который указывает указатель мыши. Этот цвет мы присвоим фону метки (`BackColor`). Код приведен в листинге 7.6.

**Листинг 7.6. Получение цвета любой точки экрана**

```
using System.Runtime.InteropServices;

private Bitmap myBitmap;

[DllImport("user32.dll", EntryPoint = "GetDC")]
public static extern IntPtr GetDC(IntPtr hWnd);

[DllImport("user32.dll", EntryPoint = "GetDesktopWindow")]
public static extern IntPtr GetDesktopWindow();

[DllImport("gdi32.dll", EntryPoint = "CreateCompatibleDC")]
public static extern IntPtr CreateCompatibleDC(IntPtr hdc);
```

```
[DllImport("user32.dll", EntryPoint = "GetSystemMetrics")]
public static extern int GetSystemMetrics(int nIndex);

[DllImport("gdi32.dll", EntryPoint = "CreateCompatibleBitmap")]
public static extern IntPtr CreateCompatibleBitmap(IntPtr hdc,
int nWidth, int nHeight);

[DllImport("gdi32.dll", EntryPoint = "SelectObject")]
public static extern IntPtr SelectObject(IntPtr hdc, IntPtr hgdiobjBmp);

[DllImport("gdi32.dll",EntryPoint="BitBlt")]
public static extern bool BitBlt(IntPtr hdcDest,int nXDest,
int nYDest,int nWidth,int nHeight,IntPtr hdcSrc,
int nXSrc,int nYSrc,int dwRop);

[DllImport("gdi32.dll",EntryPoint="DeleteDC")]
public static extern IntPtr DeleteDC(IntPtr hdc);

[DllImport("user32.dll", EntryPoint = "ReleaseDC")]
public static extern IntPtr ReleaseDC(IntPtr hWnd, IntPtr hDC);

[DllImport("gdi32.dll",EntryPoint="DeleteObject")]
public static extern IntPtr DeleteObject(IntPtr hObject);


public static Bitmap GetScreenColor()
{
        int screenX;
        int screenY;
        IntPtr hBmp;
        IntPtr  hdcScreen = GetDC(GetDesktopWindow());
        IntPtr hdcCompatible = CreateCompatibleDC(hdcScreen);

        screenX = GetSystemMetrics(0);
        screenY = GetSystemMetrics(1);
        hBmp = CreateCompatibleBitmap(hdcScreen, screenX, screenY);

        if (hBmp!=IntPtr.Zero)
        {
            IntPtr hOldBmp = (IntPtr) SelectObject(hdcCompatible, hBmp);
            BitBlt(hdcCompatible, 0, 0,screenX,screenY,
                   hdcScreen, 0, 0,13369376);
```

```
            SelectObject(hdcCompatible, hOldBmp);
            DeleteDC(hdcCompatible);
            ReleaseDC(GetDesktopWindow(), hdcScreen);

            Bitmap bmp = System.Drawing.Image.FromHbitmap(hBmp);

            DeleteObject(hBmp);
            GC.Collect();

            return bmp;
        }

        return null;
}

private void label1_MouseDown(object sender, MouseEventArgs e)
{
    myBitmap = GetDesktop();
}

private void label1_MouseUp(object sender, MouseEventArgs e)
{
    Color myColor = myBitmap.GetPixel(MousePosition.X, MousePosition.Y);
    label1.BackColor = myColor;
}
```

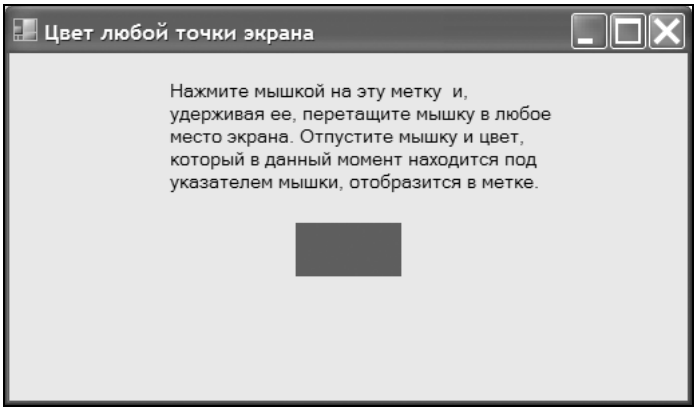



**Рис. 7.1.** Получение цвета в любой точке экрана

Запустите проект. Нажмите мышкой на метке `label1` и, удерживая ее, перетащите мышку в любое место экрана. Отпустите мышку, и цвет, который в данный момент находится под указателем мышки, отобразится в этой метке.

***Примечание***

Пример находится в папке GetScreenColorDemo на прилагаемом диске.

# Как нарисовать прямоугольник с закругленными краями?

В составе .NET Framework есть много графических методов, рисующих прямоугольники, овалы, сектора и т. п. Но, к сожалению, встроенными методами нельзя нарисовать прямоугольник с закругленными краями. Чтобы восполнить этот недостаток, мы напишем свой собственный метод `DrawRoundedRectangle` (листинг 7.7), рисующий подобный прямоугольник (рис. 7.2).

**Листинг 7.7. Создание метода, рисующего прямоугольник с закругленными краями**

```
using System.Drawing.Drawing2D;

public void DrawRoundedRectangle(Graphics g, Pen p, float X, float Y,
                                 float width, float height, float radius)
{
    GraphicsPath gp = new GraphicsPath();

    gp.AddLine(X + radius, Y, X + width - (radius * 2), Y);
    gp.AddArc(X + width - (radius * 2), Y, radius * 2,
              radius * 2, 270, 90);
    gp.AddLine(X + width, Y + radius, X + width,
              Y + height - (radius * 2));
    gp.AddArc(X + width - (radius * 2), Y + height - (radius * 2),
              radius * 2, radius * 2, 0, 90);
    gp.AddLine(X + width - (radius * 2), Y + height, X + radius,
              Y + height);
    gp.AddArc(X, Y + height - (radius * 2), radius * 2,
              radius * 2, 90, 90);
```

```
    gp.AddLine(X, Y + height - (radius * 2), X, Y + radius);
    gp.AddArc(X, Y, radius * 2, radius * 2, 180, 90);
    gp.CloseFigure();

    g.DrawPath(p, gp);
    gp.Dispose();
}

private void button1_Click(object sender, EventArgs e)
{
    DrawRoundedRectangle(CreateGraphics(), new Pen(Color.Red),
                         20, 20, 200, 200, 30);
}
```

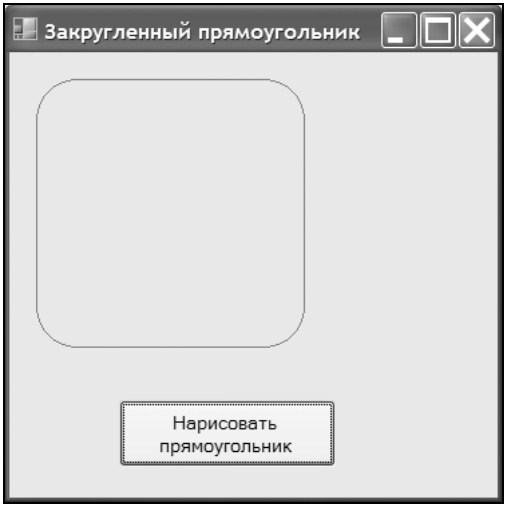



**Рис 7.2.** Рисование прямоугольника с закругленными краями

***Примечание***

Пример рисования прямоугольника с закругленными краями находится в папке DrawRoundedRectangleDemo на прилагаемом диске.

# Установка фонового изображения

В некоторых случаях желательно прятать картинки и другие файлы в ресурсах программы. Например, это позволит избежать случайного удаления кар-

тинки из папки самим пользователем. Для добавления картинки в проект в качестве ресурса необходимо проделать следующие шаги. Подготовьте графический файл, скопируйте его в папку проекта. Далее, щелкните правой кнопкой по названию проекта в окне **Solution Explorer**, выберите пункт **Add Exiting Item**, выберите нужный вам файл и нажмите на кнопку **Add**. После того как картинка добавлена в проект, нужно указать, что она должна использоваться в виде встроенного ресурса. Для этого нужно щелкнуть правой кнопкой мыши на значке графического файла в окне **Solution Explorer** и выбрать команду **Properties**. В разделе **Build Action** по умолчанию используется пункт **Content**. Измените это свойство на **Embedded Resource**. Теперь картинка является частью сборки, и для распространения программы вам понадобится единственный исполняемый файл. Чтобы установить растровое фоновое изображение для элемента управления при помощи картинки, зашитой в ресурсах программы, нужно воспользоваться кодом, приведенным в листинге 7.8.

**Листинг 7.8. Фоновый рисунок из ресурса**

```
private void Form1_Load(object sender, EventArgs e)
{
    Bitmap pic = new Bitmap(this.GetType(), "p_cat2s.jpg");

    this.button1.BackgroundImage = pic;
}
```

В этом примере мы украшаем кнопку, присвоив ее свойству `BackgroundImage` изображение из ресурсов программы.

### *Примечание*

Если вы портируете код для .NET Compact Framework, то должны постоянно помнить, что зачастую там нет такой возможности. Например, для загрузки из ресурсов приходится использовать другой код, приведенный в листинге 7.9.

**Листинг 7.9. Альтернативный способ для мобильных устройств**

```
// Загрузка растрового изображения из ресурса в .NET Compact Framework
Assembly assem = Assembly.GetExecutingAssembly();
string filename = "MyClass.mypic.bmp";
Bitmap mypic = new Bitmap(assem.GetManifestResourceStream(filename));
```

***Примечание***

Пример использования картинок из ресурсов находится в папке FromResource на прилагаемом диске.

# Как сделать снимок экрана?

Достаточно часто требуется сделать скриншот экрана или какой-нибудь его части. В старых проектах для этой цели приходилось использовать несколько функций Windows API. С выходом .NET Framework 2.0 эта задача слегка упростилась. В составе класса `Graphics` появился новый метод `CopyFromScreen`, который позволяет получить снимок экрана. Его использование показано в листинге 7.10.

**Листинг 7.10. Получение снимка экрана**

```
private void butScreenCapture_Click(object sender, EventArgs e)
{
    // Вычисляем размеры экрана
    Bitmap screenPicture = new Bitmap(Screen.PrimaryScreen.Bounds.Width,
         Screen.PrimaryScreen.Bounds.Height);
    using (Graphics g = Graphics.FromImage(screenPicture))
    {
        // копируем изображение всего экрана
        g.CopyFromScreen(0, 0, 0, 0, screenPicture.Size);
    }
    pictureBox1.Image = screenPicture;
}
```

# Сохранить изображение элемента управления или формы

Если нужно сохранить изображение не всего экрана, а только определенного элемента управления или формы, то проще воспользоваться новым методом .NET Framework 2.0 `DrawToBitmap`. Как это делать, показано в листинге 7.11.

**Листинг 7.11. Сохранение части экрана**

```
Bitmap bitmap = new Bitmap(this.Width, this.Height);
this.DrawToBitmap(bitmap,
        new System.Drawing.Rectangle(new Point(0, 0), this.Size));
// сохраняем в файл
bitmap.Save("c:\\t.jpg", System.Drawing.Imaging.ImageFormat.Jpeg);
```

# Как получить прокручиваемый рисунок?

В рассмотренном чуть выше примере, где мы работали с целым экраном, мы получили снимок экрана и поместили его в графический рисунок `PictureBox`. Но `PictureBox` не имеет встроенного свойства `AutoScroll`, поэтому мы увидим изображение экрана не целиком. И здесь мы пойдем на небольшую хитрость. Поместим на форму элемент управления `Panel` с установленным свойством `AutoScroll = true`, а свойству `SizeMode` для `PictureBox` присвоим значение `AutoSize`. Снова запустим предыдущий пример с описанными здесь изменениями. Теперь вы получите прокручиваемый рисунок и возможность увидеть весь снимок экрана.

> ***ПРИМЕЧАНИЕ***
>
> Пример с получением копий экрана находится в папке CaptureScreen на прилагаемом диске.

# Получение негатива изображения

Чтобы получить негатив изображения, нужно проделать несколько манипуляций с атрибутами изображения и матрицей преобразования (листинг 7.12).

**Листинг 7.12. Получение негатива изображения**

```
private void button1_Click(object sender, EventArgs e)
{
    // Получить изображение
    Image img = pictureBox1.Image;

    // Создаем объект атрибутов изображения
    ImageAttributes ia = new ImageAttributes();
```

```
    // создаем атрибуты по матрице преобразования
    ColorMatrix cm = new ColorMatrix();
    cm.Matrix00 = cm.Matrix11 = cm.Matrix22 = -1;

    ia.SetColorMatrix(cm);

    // Рисуем новое изображение с помощью преобразования
    Graphics g = Graphics.FromImage(img);
    g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height),
                0, 0, img.Width, img.Height, GraphicsUnit.Pixel, ia);

  // Обновить рисунок
  pictureBox1.Invalidate();

  g.Dispose();
}
```

# Сделать изображение серым

Другой эффект, который можно применить к изображению — сделать картинку серой, как на старых черно-белых фотографиях, проиллюстрирован в листинге 7.13.

**Листинг 7.13. Создание серого изображения**

```
private void button2_Click(object sender, EventArgs e)
{
    // Получить изображение
    Image img = pictureBox1.Image;

    // Создаем объект атрибутов изображения
    ImageAttributes ia = new ImageAttributes();

    // создаем атрибуты по матрице преобразования
    ColorMatrix cm = new ColorMatrix();
    cm.Matrix00 = 1 / 3f;
    cm.Matrix01 = 1 / 3f;
    cm.Matrix02 = 1 / 3f;
    cm.Matrix10 = 1 / 3f;
    cm.Matrix11 = 1 / 3f;
```

```
    cm.Matrix12 = 1 / 3f;
    cm.Matrix20 = 1 / 3f;
    cm.Matrix21 = 1 / 3f;
    cm.Matrix22 = 1 / 3f;

    ia.SetColorMatrix(cm);

    // Рисуем новое изображение с помощью преобразования
    Graphics g = Graphics.FromImage(img);
    g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height),
                0, 0, img.Width, img.Height, GraphicsUnit.Pixel, ia);

    // Обновляем рисунок
    pictureBox1.Invalidate();

    g.Dispose();
}
```

# Как создать затемненную картинку

Еще один эффект — создание затемненной картинки, как это происходит при выключении компьютера, — показан в листинге 7.14.

**Листинг 7.14. Создание затемненной картинки**

```
private void button3_Click(object sender, EventArgs e)
{
    // Получить изображение
    Image img = pictureBox1.Image;

    // Создаем объект атрибутов изображения
    ImageAttributes ia = new ImageAttributes();

    // создаем атрибуты по матрице преобразования
    ColorMatrix cm = new ColorMatrix();
    cm.Matrix00 = 0;
    cm.Matrix11 = 0;
    cm.Matrix22 = 0;
    cm.Matrix33 = 0.25f;
```

```
    ia.SetColorMatrix(cm);

    // Рисуем новое изображение с помощью преобразования
    Graphics g = Graphics.FromImage(img);
    g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height),
                0, 0, img.Width, img.Height, GraphicsUnit.Pixel, ia);

    // Обновляем рисунок
    pictureBox1.Invalidate();

    g.Dispose();
}
```

Кстати, если еще раз нажмете на кнопку, то изображение еще больше затемнится. Таким образом можно создать эффект постепенного затемнения изображения вплоть до появления черного прямоугольника.

# Эффект недоступной кнопки

Есть еще один способ получения серого изображения. У класса `ControlPaint` есть метод `DrawImageDisabled`, который используется для получения изображения недоступной кнопки. Возьмем этот метод на вооружение и напишем использующий его код (листинг 7.15).

**Листинг 7.15. Получение недоступного изображения**

```
private void button4_Click(object sender, EventArgs e)
{
    Graphics g = pictureBox1.CreateGraphics();
    ControlPaint.DrawImageDisabled(g, pictureBox1.Image,
                                   2, 2, Color.Red);
    g.Dispose();
}
```

***Примечание***

Примеры применения эффектов к изображениям находятся в папке ImageFX на прилагаемом диске.

# Как нарисовать вдавленный и выпуклый текст?

Чтобы текст на форме выглядел более красивым, его стараются сделать объемным. Существует два вида объемности — выпуклый (рис. 7.3) и вдавленный (рис. 7.4). Для достижения подобных эффектов достаточно сместить второй такой же текст относительно первого, но отобразить его другим цветом, который будет имитировать тень. Этот способ широко используется во многих графических редакторах. Нам достаточно вызвать два раза метод `DrawString` с разными смещениями (листинг 7.16).

**Листинг 7.16. Создание вдавленного и выпуклого текста**

```
using System.Drawing.Drawing2D;

private void button2_Click(object sender, EventArgs e)
{
    // Рисуем вдавленный текст
    SolidBrush solBrush = new SolidBrush(Color.Gray);
    Font f = new Font("Tahoma", 48, FontStyle.Bold);
    Graphics g = CreateGraphics();
    g.Clear(BackColor);
    g.DrawString("C#.Народные советы", f, solBrush, 10, 10);
    solBrush.Color = Color.White;
    // Смещаем строку чуть ниже и правее
    g.DrawString("C#.Народные советы", f, solBrush, 12, 11);
    g.Dispose();
}

private void button1_Click(object sender, EventArgs e)
{
    // Рисуем выпуклый текст
    SolidBrush solBrush = new SolidBrush(Color.Gray);
    Font f = new Font("Tahoma", 48, FontStyle.Bold);
    Graphics g = CreateGraphics();
    g.Clear(BackColor);
    g.DrawString("C#.Народные советы", f, solBrush, 10, 10);
    solBrush.Color = Color.White;
    // Смещаем строку чуть выше и левее
    g.DrawString("C#.Народные советы", f, solBrush, 8, 8);
    g.Dispose();
}
```

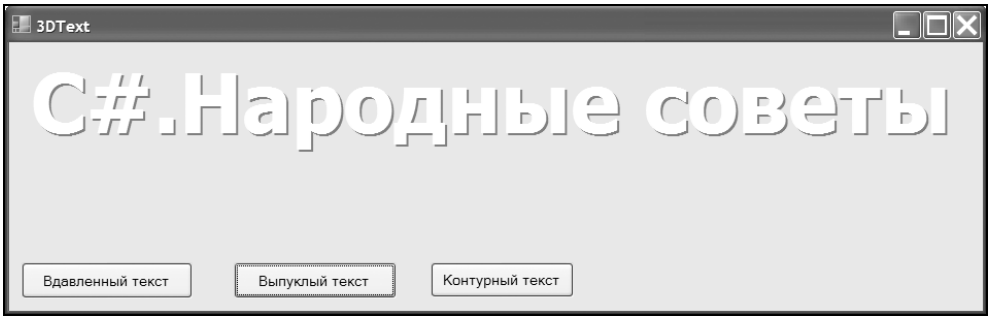



**Рис. 7.3.** Выпуклый текст

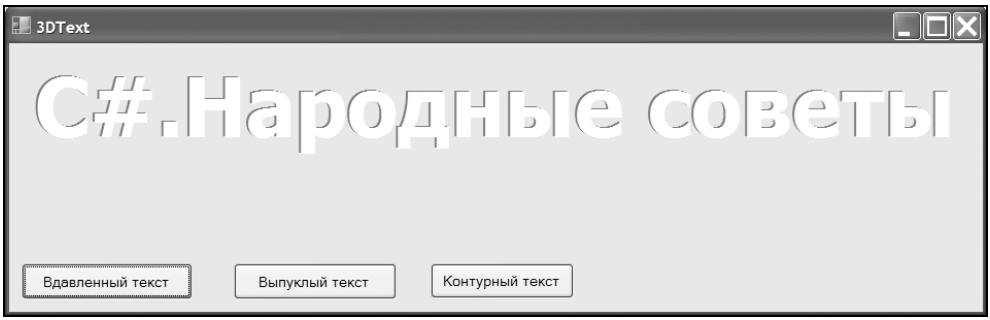



**Рис. 7.4.** Вдавленный текст

# Как получить контурный текст

Текст, выводимый на экран, по умолчанию заполнен каким-то цветом. Но можно вывести только контур символов при помощи метода `DrawPath` класса `GraphicsPath` (рис. 7.5). Данный класс позволяет создавать различные контуры из фигур и текстов, пример чего представлен в листинге 7.17.

**Листинг 7.17. Получение контурного текста**

```
private void button3_Click(object sender, EventArgs e)
{
    // Выводим контуры текста
    GraphicsPath pth = new GraphicsPath();
    // Добавляем строчку
    pth.AddString("C#.Народные советы", new FontFamily("Tahoma"),
```

```
            0, 60, new Point(30, 30),
            StringFormat.GenericDefault);

        // Создаем синее перо
        Pen p = new Pen(Color.Blue, 2);

        // Выводим контурный текст
        Graphics g = CreateGraphics();
        g.DrawPath(p, pth);

        pth.Dispose();
        g.Dispose();
}
```

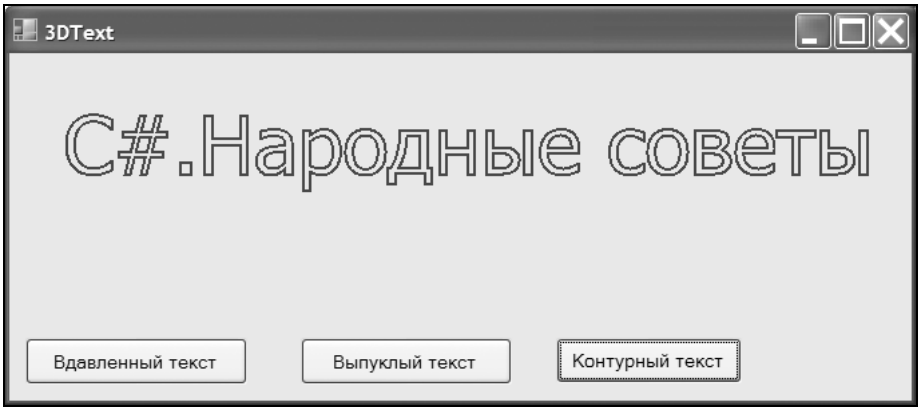



**Рис. 7.5.** Создание контурного текста

***Примечание***

Примеры создания объемных текстов находятся в папке 3DText на прилагаемом диске.

# Как отразить текст в зеркальном отражении?

Класс `Graphics` предоставляет широкие возможности для работы с графикой. Достаточно одной строчки кода, чтобы повернуть текст на любой угол, отра-

зить в зеркальном отражении по горизонтали или вертикали, наклонить символы и т. д. Как легко можно повернуть строку на 180 градусов при помощи метода `RotateTransform`, показано в листинге 7.18.

**Листинг 7.18. Отображение текста в зеркальном отражении**

```
private void button1_Click(object sender, EventArgs e)
{
    Graphics g = CreateGraphics();
    SolidBrush solBrush = new SolidBrush(Color.Gray);
    Font f = new Font("Tahoma", 32, FontStyle.Bold);

    // Разворачиваем текст на 180 градусов
    g.RotateTransform(180);
    g.DrawString("C#.Народные советы", f, solBrush, -700, -100);
    g.Dispose();
}
```

# Как повернуть текст под некоторым углом?

Соответственно, для небольшого поворота достаточно указать требуемое значение угла поворота. В листинге 7.19 мы повернем текст на 30 градусов.

**Листинг 7.19. Поворот текста на 30 градусов**

```
private void button2_Click(object sender, EventArgs e)
{
    Graphics g = CreateGraphics();

    SolidBrush solBrush = new SolidBrush(Color.Gray);
    Font f = new Font("Tahoma", 26, FontStyle.Bold);

    // Разворачиваем текст на 30 градусов
    g.RotateTransform(30);
    g.DrawString("C#.Народные советы", f, solBrush, 40, 10);
    g.Dispose();
}
```

# Вот новый поворот (из песни группы "Машина времени")

Не могу избавиться от искушения вывести текст вокруг какой-нибудь точки, чтобы получить эффект солнышка (рис. 7.6). Это делает код в листинге 7.20.

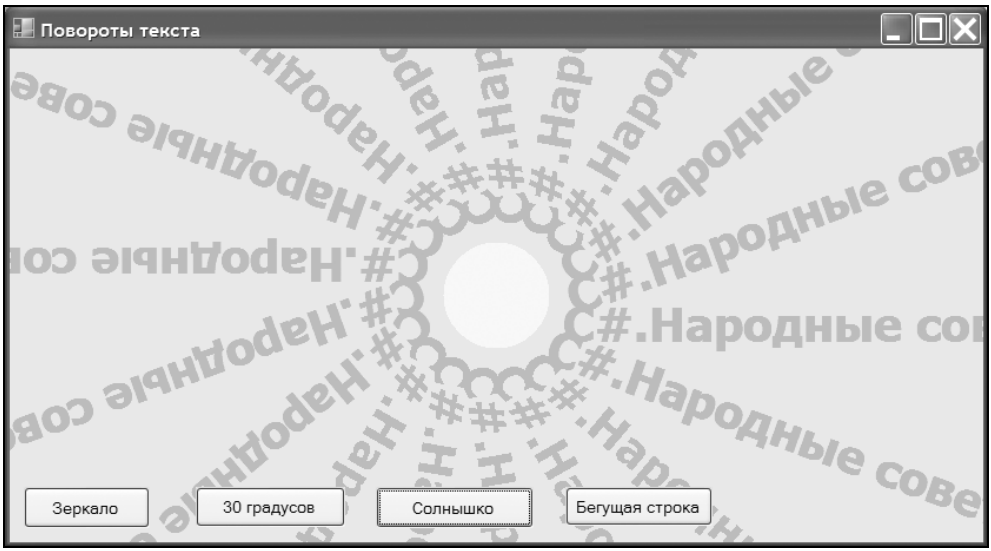



**Рис. 7.6.** Поворот текста вокруг одной точки

**Листинг 7.20. Вывод текста вокруг точки**

```
private void button3_Click(object sender, EventArgs e)
{
    Graphics g = CreateGraphics();
    g.Clear(BackColor);

    // Находим центр формы
    int cx = this.ClientSize.Width/2;
    int cy = this.ClientSize.Height/2;

    // Перемещаемся в заданную точку отсчета
    g.TranslateTransform(cx,cy);
    g.FillEllipse(Brushes.Yellow, -45,-45,90,90);
```

```
    SolidBrush solBrush = new SolidBrush(Color.YellowGreen);
    Font f = new Font("Tahoma", 24, FontStyle.Bold);

    // Рисуем солнышко
    int counter = 0;
    for(counter =0;counter<359;counter++)
    {
        g.DrawString("C#.Народные советы", f, solBrush, 50, 0);
        g.RotateTransform(20);
    }

    // Восстанавливаем преобразовние
    g.ResetTransform();
    g.Dispose();
}
```

# Бегущая градиентная строка

Мы уже рассматривали пример бегущей строки для заголовков формы. Эту же технику можно использовать для создания бегущей градиентной строки с использованием графических методов (листинг 7.21).

**Листинг 7.21. Создание бегущей градиентной строки**

```
Color startColor = Color.Green;
Color endColor = Color.Gold;
string scrollText = "C#.Народные советы        ";

private void button4_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;
}

private void timer1_Tick(object sender, EventArgs e)
{
    scrollText = scrollText.Substring(1,
            (scrollText.Length - 1)) + scrollText.Substring(0, 1);
    Invalidate();
}
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Brush grBrush = new LinearGradientBrush(this.ClientRectangle,
        startColor, endColor, 10);
    Font f = new Font(Font.Name, 60, Font.Style, GraphicsUnit.Pixel);
    e.Graphics.DrawString(scrollText,f,grBrush,0,0);
    grBrush.Dispose();
    f.Dispose();
}
```

Обратите внимание, что цвет букв во время движения плавно изменяется.

> ***ПРИМЕЧАНИЕ***
>
> Примеры отображения текстов находятся в папке RotatedText на прилагаемом диске.

# Скроллинг текста

На самом деле пример для бегущего текста получился не самым удачным. У метода `DrawString` имеются параметры, задающие координаты вывода текста. Воспользуемся этими координатами для создания скроллинга текста как по горизонтали, так и по вертикали (листинг 7.22). Для примера нам понадобится таймер и два переключателя.

**Листинг 7.22. Скроллинг текста в двух направлениях**

```
Single y_vert;
Single x_horiz;

Graphics g;
Font f;
string scrollText = "C#.Народные советы";

private void timer1_Tick(object sender, EventArgs e)
{
    if (rbHoriz.Checked)
    {
        g.Clear(this.BackColor);
```

```
        g.DrawString(scrollText, f, Brushes.Black, x_horiz, y_vert);
        if (x_horiz <= -100)
        {
            x_horiz = this.Width;
        }
        else
        {
            x_horiz -= 5;
        }
    }
    else
    {
        g.Clear(this.BackColor);
        g.DrawString(scrollText, f, Brushes.Black, x_horiz, y_vert);
        if (y_vert <= (0 - 10))
        {
            y_vert = this.Height;
        }
        else
        {
            y_vert -= 5;
        }
    }
}

private void Form1_Load(object sender, EventArgs e)
{
    g = this.CreateGraphics();
    y_vert = this.Height;
    x_horiz = this.Width - 300;

    timer1.Interval = 100;
    timer1.Start();
    f = new Font("Tahoma", 14, FontStyle.Bold, GraphicsUnit.Point);
}
```

### *Примечание*

Пример находится в папке ScrollText на прилагаемом диске.

# Анимированные картинки

.NET Framework поддерживает работу с анимированными картинками формата GIF. В частности, можно получить число кадров анимации или приостановить анимацию в любой момент. Пример приведен в листинге 7.23.

**Листинг 7.23. Работа с анимированной картинкой**

```
// Путь к анимированной картинке
Bitmap animatedImage = new Bitmap(Application.StartupPath +
        "\\" + "anicat.gif");

bool currentlyAnimating = false;

// Метод для анимации
public void AnimateImage()
{
    if (!currentlyAnimating)
    {
        ImageAnimator.Animate(animatedImage,
            new EventHandler(this.OnFrameChanged));
        currentlyAnimating = true;
    }
}

private void OnFrameChanged(object o, EventArgs e)
{
    //Force a call to the Paint event handler.
    this.Invalidate();
}

protected override void OnPaint(PaintEventArgs e)
{
    //Начинаем анимацию
    AnimateImage();

    // Получим следующий кадр
    ImageAnimator.UpdateFrames();

    // Выводим следующий кадр для анимации
    e.Graphics.DrawImage(this.animatedImage, new Point(0, 0));
}
```

```
private void button1_Click(object sender, EventArgs e)
{
   if (ImageAnimator.CanAnimate(animatedImage) == false)
        MessageBox.Show("Это не анимированная картинка");
    else
    {
        // Подсчитываем число фреймов в анимированной картинке
        FrameDimension frameDim =
            new FrameDimension(animatedImage.FrameDimensionsList[0]);
        int frameCount = animatedImage.GetFrameCount(frameDim);
        MessageBox.Show("Картинка содержит: " + frameCount + " кадров");

        // Сохраняем фреймы в отдельные файлы BMP
        for (int i = 0; i < frameCount; i++)
        {
            animatedImage.SelectActiveFrame(frameDim, i);
            MemoryStream ms = new MemoryStream();
            animatedImage.Save(ms, ImageFormat.Bmp);
            Image saveImg = Image.FromStream(ms);
            saveImg.Save(string.Format("anicat{0}.bmp", i));
        }
    }
}

private void button2_Click(object sender, EventArgs e)
{
    // Останавливаем анимацию
    ImageAnimator.StopAnimate(animatedImage,
        new EventHandler(this.OnFrameChanged));
}
```

***Примечание***

Пример работы с анимацией находится в папке Animate на прилагаемом диске.

# Как сохранить изображение из буфера обмена в файл

Пользователи часто в своей практике используют операцию копирования изображения в буфер обмена с последующей вставкой из него. Рассмотрим

пример сохранения изображения, хранящегося в буфере обмена, в файл формата JPG (листинг 7.24).

**Листинг 7.24. Сохранение изображения из буфера обмена в файл**

```
private void butSaveJPG_Click(object sender, EventArgs e)
{
    IDataObject oDataOjb =
                  System.Windows.Forms.Clipboard.GetDataObject();
    if(oDataOjb.GetDataPresent(System.Windows.Forms.DataFormats.Bitmap))
    {
        Image oImgOjb = (Image)oDataOjb.GetData(DataFormats.Bitmap,true);
        // Сохраняем в файл типа JPG
        oImgOjb.Save(@"c:\test.jpg",
                     System.Drawing.Imaging.ImageFormat.Jpeg);
    }
}
```

Для проверки работы примера запустите созданное приложение. Затем нажмите на клавишу <Print Screen> на вашей клавиатуре, чтобы скопировать изображение вашего экрана в буфер обмена. Далее нажмите на кнопку в программе, чтобы сохранить это изображение в файл. В результате ваших действий на диске C: должен появиться файл test.jpg, содержащий картинку вашего экрана. Кстати, еще раз подивитесь, как легко стало сохранять графические файлы в популярных графических форматах при помощи .NET Framework.

> ***ПРИМЕЧАНИЕ***
>
> Пример с буфером обмена находится в папке ClipBoard на прилагаемом диске.

# Шрифты и печать

## Получение списка установленных шрифтов

Чтобы получить список установленных в системе шрифтов, можно воспользоваться классом `InstalledFontCollection`, который входит в пространство имен `System.Drawing.Text`. В примере (листинг 7.25) мы проходим через все установленные шрифты и выводим их названия, используя объекты `Label`,

которые динамически создаются при получении информации о новом обнаруженном шрифте. Для удобства я присвоил свойству формы `AutoScroll` значение `True`, так как шрифтов в системе, как правило, очень много, и они не поместятся на маленькой форме.

**Листинг 7.25. Получение списка установленных шрифтов**

```
using System.Drawing.Text;

private void Form1_Load(object sender, EventArgs e)
{
    // Создаем коллекцию шрифтов
    using (InstalledFontCollection fontFamilies =
                          new InstalledFontCollection())
    {
         int offset = 10;

        // Проходим через все шрифты
        foreach (FontFamily family in fontFamilies.Families)
        {
            try
            {
                // Создаем метку, которая будет отображать текст
                // с выбранным шрифтом
                Label lblFont = new Label();
                lblFont.Text = family.Name;
                lblFont.Font = new Font(family, 14);
                lblFont.Left = 10;
                lblFont.Top = offset;
                // Добавляем Label на форму
                this.Controls.Add(lblFont);
                offset += 35;
            }
            catch
            {
                // Если выбранный шрифт не имеет стиль Обычный
                // (используется по умолчанию при создании объекта Font),
                // то может возникнуть ошибка. Игнорируем эту ошибку
                // при помощи конструкции try-catch
            }
        }
}
```

## Использование собственных шрифтов

Если вы хотите использовать в своем приложении нестандартный шрифт, то возрастает вероятность отсутствия такого шрифта у конечного пользователя. Вы можете предложить пользователю установить недостающий шрифт самостоятельно или самому позаботиться об этом. Но гораздо удобнее использовать временную установку шрифтов, которые будут использоваться только при запуске вашего приложения. Для этой операции в .NET Framework имеется класс `PrivateFontCollection`, который можно использовать для установки шрифтов, отсутствующих в операционной системе. В этом случае выполняется временная установка, не влияющая на коллекцию шрифтов, установленных в операционной системе. Иными словами, вы динамически добавляете шрифт в программу, не засоряя систему пользователя еще одним набором шрифтов (листинг 7.26). В этом примере я положил рядом с исполняемым файлом шрифт pussyfoot.ttf, который содержит силуэты кошек (рис. 7.7). В случае необходимости измените в исходном коде путь к вашему шрифту.

**Листинг 7.26. Использование собственных шрифтов в приложениях**

```
private void button1_Click(object sender, EventArgs e)
{
    PointF pointF = new PointF(310, 30);
    SolidBrush solidBrush = new SolidBrush(Color.Black);

    int count = 0;
    string familyName = "";

    FontFamily[] fontFamilies;

    PrivateFontCollection privateFontCollection =
                            new PrivateFontCollection();

    // Добавим свой шрифт в приложение.
    // Используйте свой путь к файлу шрифта
    privateFontCollection.AddFontFile(Application.StartupPath
        + "\\" + "pussyfoo.ttf");

    // Получим массив объектов FontFamily
    fontFamilies = privateFontCollection.Families;
```

```
    // Получим число объектов в массиве fontFamilies
    count = fontFamilies.Length;

    // Получим имя первого элемента из коллекции
    familyName = fontFamilies[0].Name;

    FontFamily ff = new FontFamily(familyName, privateFontCollection);
    Font f = new Font(ff, 56, FontStyle.Regular);

    Graphics g = CreateGraphics();
    // Выводим строку на экран
    g.DrawString("I LOVE C#", f, solidBrush, pointF);
    g.Dispose();
}
```

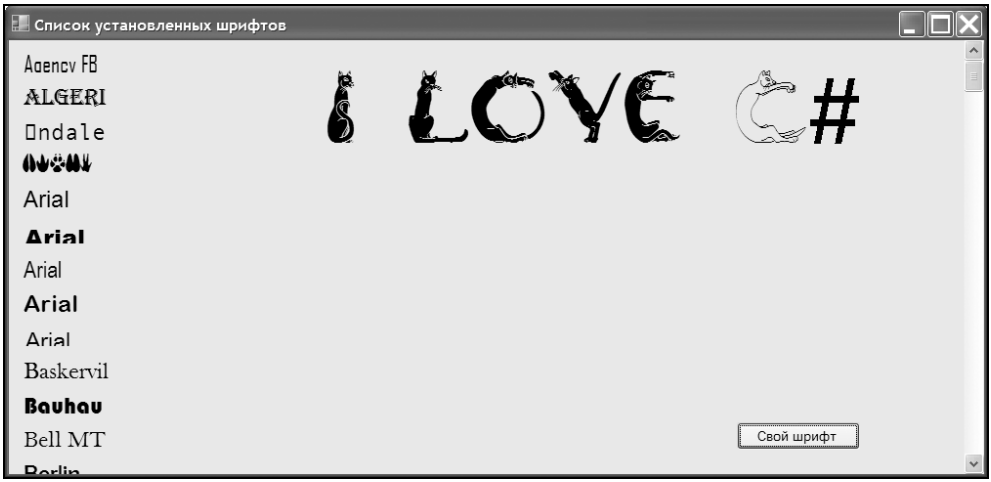



**Рис. 7.7.** Вывод на экран собственного шрифта

***ПРИМЕЧАНИЕ***

Пример находится в папке InstalledFonts на прилагаемом диске.

## Получение списка установленных принтеров

С помощью свойства `InstalledPrinters` класса `PrinterSettings` можно получить список установленных в системе принтеров и далее можно получить настройки найденных принтеров.

**Листинг 7.27. Получение списка установленных принтеров**

```
private void button1_Click(object sender, EventArgs e)
{
    foreach (string printerName in PrinterSettings.InstalledPrinters)
    {
        // Выводим имя принтера
        textBox1.Text = "Принтер: " + printerName +"\r\n";
        // Получаем настройки принтера
        PrinterSettings printer = new PrinterSettings();
        printer.PrinterName = printerName;
        // Проверяем, действителен ли принтер

        if (printer.IsValid)
        {
            // Выводим список поддерживаемых разрешений
            textBox1.Text += "Поддерживаемые разрешения:" + "\r\n";
            foreach (PrinterResolution resolution in
                                printer.PrinterResolutions)
            {
                 textBox1.Text += resolution + "\r\n";
             }

             // Выводим список доступных размеров бумаги
             textBox1.Text += "Поддерживаемые размеры бумаги:" + "\r\n";
             foreach (PaperSize size in printer.PaperSizes)
             {
                 if (Enum.IsDefined(size.Kind.GetType(), size.Kind))
                 {
                     textBox1.Text += size + "\r\n";
                 }
             }
         }
     }
 }
```

## Как распечатать документ?

Вам необходимо распечатать текст или картинку из вашего приложения? В этом случае вам нужно создать объект `PrintDocument` и использовать событие `PrintDocument.PrintPage`, чтобы работать с методами `DrawString` и

`DrawImage` для печати данных, а также сконфигурировать свойства печати. После этого вы можете вызывать метод `Print`. Настройка свойств принтера осуществляется через свойства `PrintDocument.PrinterSettings` и `PrintDocument.DefaultPageSettings`. Вы можете настраивать свойства в коде или через стандартное диалоговое окно печати. В этом окне пользователь может выбрать принтер и количество копий документа. Листинг 7.28 содержит пример печати одной страницы.

**Листинг 7.28. Печать документа**

```
using System.Drawing.Printing;
using System.IO;

private void butPrint_Click(object sender, EventArgs e)
{
    // Создаем документ и прикрепляем к нему обработчик события
    PrintDocument doc = new PrintDocument();
    doc.PrintPage += this.Doc_PrintPage;

    // Пользователь может выбирать принтер и его свойства через
    // стандартное диалоговое окно
    PrintDialog dlgSettings = new PrintDialog();
    dlgSettings.Document = doc;

    // Если выбрана кнопка OK, то печатаем документ
    if (dlgSettings.ShowDialog() == DialogResult.OK)
    {
        doc.Print();
    }
}

private void Doc_PrintPage(object sender, PrintPageEventArgs e)
{
    // Задаем шрифт
    using (Font font = new Font("Arial", 30))
    {
        // Определяем позиции для печати данных
        float x = e.MarginBounds.Left;
        float y = e.MarginBounds.Top;

        float lineHeight = font.GetHeight(e.Graphics);
```

```
        // Печатаем пять строчек текста
        for (int i = 1; i <= 5; i++)
        {
            e.Graphics.DrawString("Это строчка номер " + i.ToString(),
               font, Brushes.Black, x, y);
            // Смещаемся вниз для следующей строчки
            y += lineHeight;
        }

        y += lineHeight;

        // Рисуем картинку из файла
        e.Graphics.DrawImage(Image.FromFile(
            Path.Combine(Application.StartupPath, "test.jpg")), x, y);
    }
}
```

## Как показать окно предварительного просмотра перед печатью

Элемент управления `PrintPreviewDialog` является стандартным диалоговым окном, который используется для отображения компонента `PrintDocument` в том виде, как он будет напечатан. Данное окно используется в качестве простого и быстрого решения вместо диалогового окна, настраиваемого самостоятельно. В нем имеются кнопки для печати, изменения масштаба, отображения одной или нескольких страниц, а также для закрытия диалогового окна. Ключевым свойством этого элемента управления является свойство `Document`, задающее документ, который требуется просмотреть. Этот документ должен являться объектом `PrintDocument`. Чтобы вывести диалоговое окно, необходимо вызвать для него метод `Show`. Использование элемента управления `PrintPreviewDialog` иллюстрируется в листинге 7.29.

**Листинг 7.29. Вывод окна предварительного просмотра перед печатью**

```
private void butPreview_Click(object sender, EventArgs e)
{
    // Создаем документ и прикрепляем к нему обработчик события
    PrintDocument doc = new PrintDocument();

    doc.PrintPage += this.Doc_PrintPage;
```

```
    PrintPreviewDialog dlgPreview = new PrintPreviewDialog();
    dlgPreview.Document = doc;
    dlgPreview.Show();
}
```

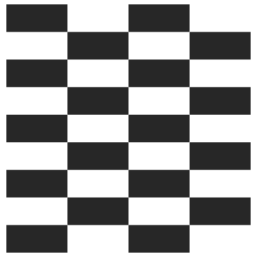
***Примечание***

Пример находится в папке InstalledPrinters на прилагаемом диске.

# Заключение

Работа с графикой — неисчерпаемая тема для разговоров. Для тех, кто всерьез увлекается графикой, могу посоветовать изучить программу Paint.NET, которая поставляется с исходными кодами. Графический редактор Paint.NET является своеобразной попыткой написать аналог такого мощного графического пакета, как Adobe Photoshop, используя лишь C#. Домашняя страница программы находится по адресу **http://www.getpaint.net/**.

# Глава 8



# Работа с мышью и клавиатурой

Мышь и клавиатура пока являются основными средствами ввода информацию в компьютер со стороны пользователя. Поэтому обойти тему работы с этими устройствами никак нельзя. Здесь вы узнаете несколько полезных советов и трюков, которые, надеюсь, пригодятся в вашей практике.

## Мышь

В современных оконных системах мышь является основным средством ввода, поэтому уметь с ней работать должен каждый программист.

### Как скрыть и показать указатель мыши?

Скрыть указатель мыши с экрана очень просто. Для этого достаточно воспользоваться методом `Cursor.Hide`. Чтобы снова показать указатель мыши на экране, надо воспользоваться противоположным методом `Cursor.Show`. Но необходимо помнить, что метод `Cursor.Hide` обладает накопительным свойством. Если вы два раза вызовете этот метод, то вам придется также два раза вызывать и метод `Cursor.Show`, восстанавливающий указатель мыши. Предположим, мы хотим скрыть указатель мыши, когда он находится над кнопкой, и показать его снова, когда он выходит за пределы кнопки. Реализация этого действия показана в листинге 8.1.

**Листинг 8.1. Скрытие и показ указателя мыши**

```
private void butTestMouse_MouseEnter(object sender, EventArgs e)
{
    // Прячем курсор, когда он находится над кнопкой
```

```
    Cursor.Hide();
}

private void butTestMouse_MouseLeave(object sender, EventArgs e)
{
    // Выводим курсор, когда он выходит за пределы кнопки
    Cursor.Show();
}
```

Как вы понимаете, эту возможность можно использовать при создании программ-шуток, чтобы немножко попугать пользователя.

## Как установить позицию указателя мыши?

Свойство `Cursor.Position` не только возвращает позицию указателя мыши, но и задает ее координаты. Поэтому с помощью данного свойства можно перемещать указатель в любую точку экрана. Необходимо помнить, что объект `Point`, используемый в этом свойстве, содержит экранные координаты. Например, чтобы переместить курсор мыши на 30 точек вниз, используйте код из листинга 8.2.

**Листинг 8.2. Устанавливаем новую позицию курсора**

```
Point pt = Cursor.Position;
pt.Y += 30;
Cursor.Position = pt;
```

Чтобы убедиться, что код работает, проделайте следующее. Поместите указатель мыши приблизительно в центре экрана и нажмите на клавишу пробела или ввода (<Enter>). Так как на форме у нас присутствует только одна кнопка, то она автоматически получит фокус и, следовательно, после нажатия на указанные клавиши выполнится код обработчика нажатия кнопки. Можете самостоятельно усложнить пример — добавьте в проект таймер и заставьте курсор беспорядочно бегать по экрану, используя случайные числа. Только не забудьте предусмотреть возможность закрытия программы, а то сумасшедший курсор не позволит вам нажать на крестик в правом углу приложения!

## Анимированные курсоры

В составе .NET Framework нет класса, позволяющего работать с анимированными курсорами ANI. Поэтому воспользуемся (листинг 8.3) вызовом функции Windows API `LoadCursorFromFile`.

**Листинг 8.3. Использование анимированного курсора**

```
[DllImport("user32.dll", EntryPoint = "LoadCursorFromFileW",
CharSet = CharSet.Unicode)]
public static extern IntPtr LoadCursorFromFile(String fileName);

private void button1_Click(object sender, EventArgs e)
{
    string fileName = @"D:\archive\cursors\cat-h.ani";
    // загружаем курсор из файла
    IntPtr hCursor = LoadCursorFromFile(fileName);
    // устанавливаем его в качестве текущего курсора формы
    this.Cursor = new Cursor(hCursor);
}
```

***Примечание***

Пример работы с анимированным курсором находится в папке AnimatedCursors на прилагаемом диске.

## Мышеловка

Если у вас есть кот, который отказывается ловить мышей, то соорудим виртуальную мышеловку, которая будет ловить ваш указатель мыши. Пусть ваш кот учится на наглядных примерах. Добавим на форму элемент `Label` и изменим его цвет, чтобы он не сливался с формой и был хорошо виден нам. У класса `Cursor` имеется свойство `Clip`, которое возвращает или задает границы, представляющие прямоугольник отсечения для курсора. Мы воспользуемся этим свойством и в качестве прямоугольника отсечения зададим поверхность надписи `Label`, как показано в листинге 8.4.

**Листинг 8.4. Создание мышеловки для курсора**

```
private void lblTrapMouse_MouseEnter(object sender, EventArgs e)
{
```

```
    // Чтобы освободить мышь из мышеловки,
    // используйте комбинацию клавиш <Ctrl>+<Alt>+<Del>
    Cursor.Clip = RectangleToScreen(new Rectangle(lblTrapMouse.Location,
                                                  lblTrapMouse.Size));
            lblTrapMouse.Text = "Попался, который кусался!";
}
```

Запустите проект и поместите указатель мыши в закрашенную область. Зря вы это сделали! Ваша мышка попалась в мышеловку. Попробуйте переместить мышь за пределы этой области. У вас ничего не получится. Вы даже можете нажать на кнопку из предыдущего примера, который перемещает курсор мыши вниз. Указатель мыши переместится вниз, но только в том случае, если он не выйдет за пределы ограничивающего прямоугольника. Вы можете переключиться на другие приложения при помощи комбинации клавиш <Alt>+<Tab>, но мышь по-прежнему будет оставаться в ловушке. Чтобы выйти из нее, вам нужно нажать комбинацию клавиш <Ctrl>+<Alt>+<Del>. Вы это теперь знаете, а знает ли это пользователь, которому вы подсунете эту программку?

> ***ПРИМЕЧАНИЕ***
>
> Пример работы с мышью находится в папке SimpleMouse на прилагаемом диске.

## Право выбора

Продолжим опыты с мышью. Мы знаем, что можем устанавливать указатель мыши в любую позицию программным путем. Используя данную особенность, создадим интересный эффект, когда у пользователя не будет возможности щелкнуть на кнопке. Поместим на форме две кнопки с надписями **Да** и **Нет**, а также разместим какой-нибудь пояснительный текст с вопросом. Предположим, это будет вопрос **Вы хотите посетить сайт http://rusproject.narod.ru?**. Естественно, мне, как автору сайта, не хотелось бы, чтобы пользователь выбирал отрицательный вариант. Когда пользователь попытается подвести курсор мыши к кнопке с надписью **Нет**, его сразу отбросит на кнопку **Да**. В листинге 8.5 показано, как это реализуется в коде.

**Листинг 8.5. Принудительное перемещение курсора**

```
private void butNo_MouseMove(object sender, MouseEventArgs e)
{
```

```
    Point myPoint = butYes.PointToScreen(new Point(0, 0));
    myPoint.X += butYes.Width / 2;
    myPoint.Y += butYes.Height / 2;
    Cursor.Position = myPoint;
}
```

Мы написали код для обработчика события `MouseMove`. Это событие возникает, как только курсор мыши достигает границ кнопки. В этом случае сразу же вычисляется местоположение другой кнопки, и курсор мыши автоматически перемещается в нужное место. Еще раз обратите внимание, что для вычисления координат кнопки мы используем метод `PointToScreen`, который конвертирует координаты из клиентской системы координат в экранную систему координат. Данный пример носит немного шутливый характер, но на самом деле этот пример можно использовать не только в качестве шутки. Иногда требуется автоматически перенаправить курсор мыши в нужное место. Давайте напишем универсальную процедуру, представленную в листинге 8.6.

**Листинг 8.6. Универсальная процедура перемещения мыши**

```
public void JumpToControl(Control ctrl)
{
    // Перемещаем курсор мыши в нижнюю часть середины элемента
    Point objPoint = ctrl.PointToScreen(new Point(0, 0));
    objPoint.X += ctrl.Width / 2;
    objPoint.Y += (ctrl.Height / 4) * 3;
    Cursor.Position = objPoint;
}

private void butYes_Click(object sender, EventArgs e)
{
    JumpToControl(lnkSite);
}
```

Как видите, мы поместили созданную процедуру в событие `Click` кнопки с надписью **Да** и указали элемент, на который должен переместиться указатель мыши. В нашем случае я разместил на форме для демонстрации примера элемент `LinkLabel` с именем `lnkSite`. Обратите внимание, что в нашей универсальной процедуре используется перемещение не в середину элемента управления, а на три четверти его высоты для большей изящности.

***Примечание***

Пример работы с указателями мыши вы найдете в папке TwoButtons на прилагаемом к книге компакт-диске.

## Меняем кнопки мыши местами

У вашей мыши имеются, как минимум, две кнопки. Если вы правша, то основная кнопка у вас левая, а правая кнопка служит для вызова контекстного меню. Но у левшей своя точка зрения на эти кнопки, и основная кнопка у них правая. Чтобы поменять настройки для кнопок мыши, пользователь должен зайти в Панель управления и в апплете **Мышь** установить желаемые настройки. Но вы можете программно изменить эти настройки (листинг 8.7) через вызов функции Windows API `SwapMouseButton`.

**Листинг 8.7. Замена кнопок мыши**

```
using System.Runtime.InteropServices;

[DllImport("user32.dll", EntryPoint = "SwapMouseButton")]
internal extern static int SwapMouseButton(int bSwap);

private void butSwapMouse_Click(object sender, EventArgs e)
{
    // Меняем кнопки мыши местами
    SwapMouseButton(1);
}

private void butRestoreMouse_Click(object sender, EventArgs e)
{
    // Восстанавливаем настройки мыши
    SwapMouseButton(0);
}
```

Чтобы убедиться, что кнопки мыши действительно поменялись, я расположил на форме текстовое поле, у которого по умолчанию имеется контекстное меню. Запустите проект и нажмите на кнопку с надписью **Поменять местами**. Теперь щелкните правой кнопкой на текстовом поле. Ничего не происходит? Так ведь правая кнопка теперь работает у вас как левая. Теперь щелкните на текстовом поле левой кнопкой — у вас появится контекстное меню. Замена кнопок местами носит глобальный характер и действует на все при-

ложения. Если хотите вернуть старые настройки, то нажмите на кнопку **Восстановить** (догадайтесь, какой кнопкой надо сделать щелчок).

***Примечание***

Пример замены кнопок мыши находится в папке AdvancedMouse на прилагаемом диске.

## Как узнать координаты мыши?

Обратите внимание, что в .NET Framework 2.0 кроме событий `Click` и `DoubleClick` появились новые события `MouseClick` и `MouseDoubleClick`, которые в отличие от старых событий позволяют получить координаты мыши в момент щелчка. Как их использовать, показано в листинге 8.8.

**Листинг 8.8. Получение координат мыши в момент щелчка**

```
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    // Выводим координаты мыши в заголовке формы
    this.Text = "MouseClick: " +  e.Button.ToString() +
                " " + e.Location.ToString();
}

private void Form1_MouseDoubleClick(object sender, MouseEventArgs e)
{
    // Выводим координаты мыши в заголовке формы
    this.Text = "MouseDoubleClick: " + e.Button.ToString() +
                " " + e.Location.ToString();
}
```

***Примечание***

Пример, в котором мы получаем координаты мыши, находится в папке MouseClick на прилагаемом диске.

## Как преобразовать экранные координаты в клиентские (для данного элемента) и наоборот?

Для преобразования координат точки используйте методы `Control.PointToClient` (преобразовывает экранные координаты в клиент-

ские) и `Control.PointToScreen` (преобразовывает клиентские координаты в экранные). Соответствующие методы существуют и для прямоугольника: `Control.RectangleToClient` и `Control.RectangleToScreen`. Чуть ранее я показывал пример с использованием этих методов (см. листинги 8.4 и 8.5).

## Как двигать указателем мыши программно?

Если мы знаем, как устанавливать указатель мыши в заданной позиции, то нам по плечу и более сложная задача. Например, если мы хотим заставить указатель мыши двигаться по замысловатой траектории, то можно воспользоваться свойством `PathData.Points`. Данное свойство позволяет получить координаты всех точек, входящих в траекторию. Таким образом можно создать траекторию в виде окружности и заставить указатель крутиться вдоль этой фигуры. Листинг 8.9 показывает, как это реализовать.

**Листинг 8.9. Управление курсором мыши программным способом**

```
using System.Drawing.Drawing2D;

/// <summary>
/// Метод, позволяющий перемещать указатель мыши
/// по заданной траектории (например, вдоль окружности)
/// </summary>
/// <param name="times">Количество повторов</param>
public void MoveMouse(int times)
{
    GraphicsPath gp = new GraphicsPath();
    // Размеры экрана
    Rectangle screensize = Screen.GetBounds(new Point(0, 0));
    // Выберем произвольный прямоугольник в области экрана
    Rectangle rect = new Rectangle((int)screensize.Width/2-100,
        (int)screensize.Height/2-100, 200, 200);
    // Добавим в траекторию окружность
    gp.AddEllipse(rect);
    // Разобьем окружность на серию линий
    gp.Flatten();

    // цикл повторений движения указателя мыши
    for (int repeat = 0; repeat <= times - 1; repeat++)
    {
        // выводим в заголовке формы число витков
```

```
        this.Text = (repeat+1).ToString();
        // проходим через все точки траектории
        for(int i=0; i <= gp.PathData.Points.Length-1; i++)
        {
            // организуем небольшую задержку
            System.Threading.Thread.Sleep(10);
            // перемещаем указатель мыши
            Cursor.Position = new Point((int)gp.PathData.Points[i].X,
                                        (int)gp.PathData.Points[i].Y);
        }
    }
    gp.Dispose();
}

private void butStart_Click(object sender, EventArgs e)
{
    MoveMouse(5);
}
```

Поступая аналогично, можно заставить курсор мыши обрисовать контуры формы. Подобные приемы управления курсором (листинг 8.10) могут пригодиться для создания презентаций.

**Листинг 8.10. Движение курсора вокруг формы**

```
public void MoveAroundForm()
{
    GraphicsPath gp = new GraphicsPath();

    gp.AddLine(this.DesktopLocation.X + 2,
        this.DesktopLocation.Y + 2,
        this.DesktopLocation.X + this.Size.Width - 2,
        this.DesktopLocation.Y - 2);

    gp.AddLine(this.DesktopLocation.X + this.Size.Width - 2,
        this.DesktopLocation.Y - 2,
        this.DesktopLocation.X + this.Size.Width - 2,
        this.DesktopLocation.Y + this.Size.Height - 2);

    gp.AddLine(this.DesktopLocation.X + this.Size.Width - 2,
        this.DesktopLocation.Y + this.Size.Height - 2,
        this.DesktopLocation.X + 2,
        this.DesktopLocation.Y + this.Size.Height + 2);
```

```
    gp.AddLine(this.DesktopLocation.X + 2,
        this.DesktopLocation.Y + this.Size.Height + 2,
        this.DesktopLocation.X + 2,
        this.DesktopLocation.Y + 2);

    int i;
    for (i = 0; i <= gp.PathData.Points.Length - 1; i++)
    {
        System.Threading.Thread.Sleep(1000);

        Cursor.Position = new Point((int)gp.PathData.Points[i].X,
            (int)gp.PathData.Points[i].Y);
    }
    gp.Dispose();

}

private void butShowBorderForm_Click(object sender, EventArgs e)
{
    MoveAroundForm();
}
```

***ПРИМЕЧАНИЕ***

Пример управления курсором мыши находится в папке MovingMouse на прилагаемом диске.

## Как выполнить эмуляцию щелчков мыши?

Чтобы эмулировать щелчки мышью, можно воспользоваться функциями Windows API `SendInput` или `mouse_event`. В листинге 8.11 приводится пример перемещения мыши в нижний левый угол экрана и нажатия правой кнопки мыши, что вызовет открытие контекстного меню кнопки **Пуск**.

**Листинг 8.11. Эмуляция щелчков мыши**

```
using System.Runtime.InteropServices;

[DllImport("user32.dll")]
static extern void mouse_event(int dwFlags,
int dx, int dy, int dwData, UIntPtr dwExtraInfo);
```

```
const int MOUSEEVENTF_MOVE = 0x0001;
const int MOUSEEVENTF_LEFTDOWN = 0x0002;
const int MOUSEEVENTF_LEFTUP = 0x0004;
const int MOUSEEVENTF_RIGHTDOWN = 0x0008;
const int MOUSEEVENTF_RIGHTUP = 0x0010;
const int MOUSEEVENTF_ABSOLUTE = 0x8000;

private void button1_Click(object sender, EventArgs e)
{
    // Перемещаем в район кнопки Пуск
    mouse_event(MOUSEEVENTF_MOVE | MOUSEEVENTF_ABSOLUTE,
                400, 65000, 0, UIntPtr.Zero);
    // Нажимаем и отпускаем правую кнопку мыши
    mouse_event(MOUSEEVENTF_RIGHTDOWN, 0, 0, 0, UIntPtr.Zero);
    mouse_event(MOUSEEVENTF_RIGHTUP, 0, 0, 0, UIntPtr.Zero);
}
```

Необходимо помнить, что кнопка **Пуск** у пользователя может находиться не только в нижнем левом углу, но и в других позициях. Свой пример я писал применительно к конкретному случаю на моем Рабочем столе.

> ***Примечание***
>
> Пример программной эмуляции нажатия кнопок мыши находится в папке APIClicks на прилагаемом диске.

## Рисование

Наверняка, вам приходилось рисовать с помощью мыши в каком-нибудь графическом редакторе, например, в стандартной утилите Paint, которая входит в состав любой операционной системы Windows. Мы попробуем написать свою программу, позволяющую выполнять эти же действия. Рассмотрим этот процесс подробнее. Рисование начинается с помещения указателя мыши на специально отведенную область для рисования. Это может быть клиентская область окна нашей программы или графическое поле. Следующий этап — нажатие кнопки мыши. Причем допускается нажатие как левой, так и правой кнопки мыши. В некоторых редакторах используются разные варианты, в зависимости от задумок разработчика. После нажатия кнопки мы перемещаем мышь в другое место, не отпуская кнопки. Во время перемещения за указателем остается след, позволяющий нам контролировать движения мы-

ши. И, наконец, последняя операция — отпускание кнопки мыши. Естественно, рисование тут же прекращается. Описанное действие можно повторять неограниченное число раз. Таким образом, рисование заключается в обработке событий мыши `MouseDown`, `MouseMove` и `MouseUp`. Вооружившись полученными знаниями, приступим к созданию простого графического редактора (рис. 8.1), код которого представлен в листинге 8.12.

**Листинг 8.12. Простейший графический редактор**

```
// Координаты мыши в момент нажатия
int x_md, y_md;
// Будем рисовать пером синего цвета и толщиной 3 пиксела
Pen p = new Pen(Color.Blue, 3);
// Отслеживаем нажатия кнопки, чтобы знать
// когда пользователь заканчивает рисовать
bool bPaint;

private void Form1_MouseDown(object sender, MouseEventArgs e)
{
    // Начинаем рисовать
    bPaint = true;
    // Координаты мыши в момент нажатия
    x_md = e.X;
    y_md = e.Y;
}

private void Form1_MouseMove(object sender, MouseEventArgs e)
{
    if (bPaint)
    {
        // Создадим объект Graphics для рисования
        Graphics g = CreateGraphics();

        // Определяем координаты мыши при перемещении
        int x_mm = e.X;
        int y_mm = e.Y;
        // Рисуем
        g.DrawLine(p, x_md, y_md, x_mm, y_mm);

        // Передаем координаты мыши в переменные класса
        x_md = x_mm;
        y_md = y_mm;
```

```
        g.Dispose();
    }
}

private void Form1_MouseUp(object sender, MouseEventArgs e)
{
    bPaint = false;
}

private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    // Очистим форму от наших художеств при нажатии правой кнопки мыши
    if (e.Button == MouseButtons.Right)
    {
        Graphics g = CreateGraphics();
        g.Clear(this.BackColor);
        g.Dispose();
    }
}
```

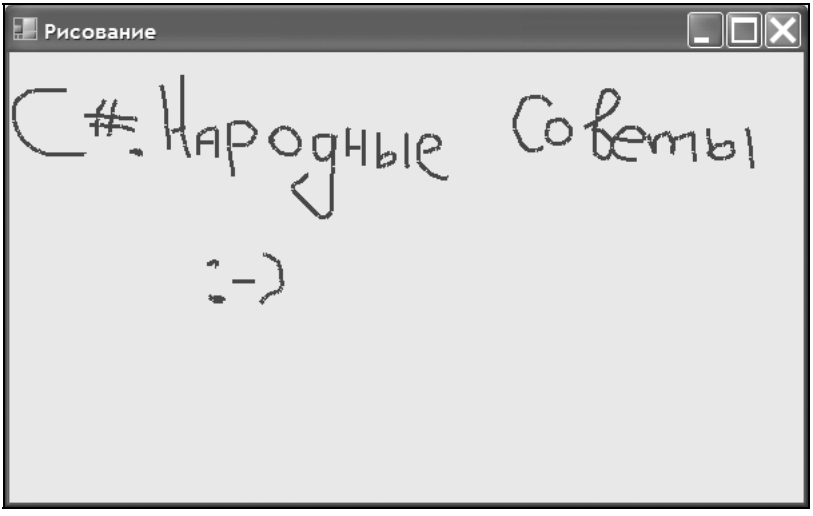



**Рис. 8.1.** Простейший графический редактор в действии

### *Примечание*

Пример простейшего графического редактора находится в папке Simple-Paint на прилагаемом диске.

# Работа с клавиатурой

Клавиатура является одной из важных составляющих компьютера. Если вы отключите клавиатуру и попробуете включить машину, то появится сообщение об ошибке и система откажется загружаться. Клавиатура позволяет вводить информацию в компьютер с помощью клавиш. Несмотря на разнообразие моделей и количества клавиш на них, у всех клавиатур имеется определенный стандарт клавиш. Прежде всего, это ряд клавиш с буквами и цифрами, клавиши-модификаторы, функциональные клавиши и цифровой блок. Также у клавиатур имеются индикаторы со светодиодами, которые могут загораться и выключаться при нажатии некоторых клавиш.

## Как переключать раскладки клавиатуры?

В отличие от американцев, нам приходится использовать две раскладки клавиатуры, а то и больше. Таким образом, у вас периодически возникает потребность переключения раскладки. Пользователь может изменить раскладку вручную, нажав комбинацию клавиш <Alt>+<Shift> (или другую комбинацию, определенную настройками системы). В последнее время появились программы, которые автоматически переключают раскладку, пытаясь определить нужный язык по вводимым словам. Но если эти программы способны программно переключать раскладки, то почему бы и нам не сделать то же самое. Для этого нужно воспользоваться свойством `CurrentInputLanguage` класса `System.Windows.Forms.InputLanguage`. В листинге 8.13 приведен код, переключающий язык ввода на русский.

**Листинг 8.13. Переключение раскладок клавиатуры**

```
using System.Globalization;

InputLanguage.CurrentInputLanguage =
                InputLanguage.FromCulture(new CultureInfo("ru-RU"));


//Если нужно переключиться на английский язык, то используйте строчку
InputLanguage.CurrentInputLanguage =
                InputLanguage.FromCulture(new CultureInfo("en-US"));
```

Если вам нужно переключиться в другую раскладку (украинскую, белорусскую или финскую), то поищите названия соответствующих раскладок в до-

кументации. Кроме того, в *главе 14* более подробно рассказано о глобализации и локализации программ, в которых используются возможности пространства имен `System.Globalization`.

## Как получить текущий язык ввода?

Мы рассмотрели только что пример, как программно переключить раскладку клавиатуры. Если вам необходимо сначала узнать текущую раскладку, то используйте код, приведенный в листинге 8.14.

**Листинг 8.14. Получение текущего языка ввода**

```
private void Form1_Load(object sender, EventArgs e)
{
    InputLanguage myDefaultLanguage = InputLanguage.DefaultInputLanguage;
    InputLanguage myCurrentLanguage = InputLanguage.CurrentInputLanguage;
    textBox1.Text = "Текущий язык ввода: " +
        myCurrentLanguage.Culture.EnglishName+ "\r\n";
    textBox1.Text += "Язык ввода по умолчанию: " +
        myDefaultLanguage.Culture.EnglishName + "\r\n";

    // выводим текущий язык ввода
    InputLanguage myCurrentLanguage2 =
        InputLanguage.CurrentInputLanguage;
    textBox1.Text += "Новый текущий язык ввода: " +
        myCurrentLanguage2.Culture.EnglishName;
}
```

***Примечание***

Пример работы с раскладками клавиатур находится в папке Keyboard на прилагаемом диске.

## Как послать нажатия клавиш программно?

С помощью класса `SendKeys`, который находится в пространстве имен `System.Windows.Forms`, можно программно эмулировать нажатие определенной комбинации клавиш активному приложению, используя его метод `SendWait`. Так как не существует управляемых методов для активирования других приложений, то для передачи фокуса другому приложению воспользуемся функциями Windows API `FindWindow` и `SetForegroundWindow`. Связка

этих функций с классом SendKeys позволит нам, например, запустить программу Блокнот и напечатать несколько слов (листинг 8.15).

**Листинг 8.15. Программные нажатия на клавиши**

```
[DllImport("USER32.DLL")]
static extern IntPtr FindWindow(string lpClassName, string lpWindowName);

[DllImport("USER32.DLL")]
static extern bool SetForegroundWindow(IntPtr hWnd);

private void Form1_Load(object sender, EventArgs e)
{
     Process note = Process.Start("notepad.exe");
     note.WaitForInputIdle();

     IntPtr notepadHandle = FindWindow(null, "Блокнот");

     // Другой вариант
     // IntPtr notepadHandle = FindWindow("Notepad", null);

     SetForegroundWindow(notepadHandle);
     SendKeys.SendWait("C# ");
     SendKeys.SendWait(" Народные советы");
     SendKeys.SendWait("~");
     SendKeys.SendWait("    Обалдеть!");
}
```

При запуске программы автоматически запустится новая копия программы Блокнот, в которой будут напечатаны нужные слова.

> ***Примечание***
>
> Пример программного нажатия на клавиши находится в папке SendKeysDemo на прилагаемом диске.

## Как включать и выключать индикаторы клавиш <Caps Lock>, <Num Lock> и <Scroll Lock>?

На большинстве клавиатур имеются светодиоды-индикаторы, которые позволяют пользователю отслеживать состояние клавиш <Caps Lock>,

<Num Lock> и <Scroll Lock>. Например, если горит индикатор Caps Lock, значит, клавиша <Caps Lock> активирована, и все символы, вводимые в текстовом поле, будут выводиться в верхнем регистре (если при этом удерживать клавишу <Shift>, то символы будут выводиться в нижнем регистре). Мы можем программно изменить состояние этих клавиш и устроить небольшую светомузыку, включая и выключая индикаторы в такт какой-нибудь мелодии. Для реализации этой идеи нам понадобится вызов функций Windows API `keybd_event`, как показано в листинге 8.16.

**Листинг 8.16. Включение индикаторов на клавиатуре**

```
using System.Runtime.InteropServices;

[DllImport("user32.dll")]
static extern void keybd_event(byte bVk, byte bScan, uint dwFlags,
UIntPtr dwExtraInfo);

private const int VK_NUMLOCK = 0x90;
private const int VK_SCROLL = 0x91;
private const int VK_CAPITAL = 0x14;
private const int KEYEVENTF_EXTENDEDKEY = 0x1;
private const int KEYEVENTF_KEYUP = 0x2;

private void butCapsLock_Click(object sender, EventArgs e)
{
    // Включаем индикатор Caps Lock
    keybd_event(VK_CAPITAL, 0x45, KEYEVENTF_EXTENDEDKEY, (UIntPtr)0);
    keybd_event(VK_CAPITAL, 0x45,
                KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, (UIntPtr)0);
}

private void butNumLock_Click(object sender, EventArgs e)
{
    // Включаем или выключаем индикатор Num Lock
    keybd_event(VK_NUMLOCK, 0x45, KEYEVENTF_EXTENDEDKEY, (UIntPtr)0);
    keybd_event(VK_NUMLOCK, 0x45,
                KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, (UIntPtr)0);
}

private void butScrollLock_Click(object sender, EventArgs e)
{
```

```
    // Включаем или выключаем индикатор Scroll Lock
    keybd_event(VK_SCROLL, 0x45, KEYEVENTF_EXTENDEDKEY, (UIntPtr)0);
    keybd_event(VK_SCROLL, 0x45,
                KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, (UIntPtr)0);
}
```

Как видите, для включения какой-нибудь клавиши с индикатором нужно дважды вызвать функцию `keybd_event`. Повторный двойной вызов этой функции выключает индикатор, поэтому нет необходимости писать какой-либо дополнительный код для этого.

## Как определить состояние клавиш-индикаторов?

У вас может возникнуть вопрос, а как узнать текущее состояние этих клавиш-индикаторов перед их включением или выключением. В .NET Framework 1.1 приходилось использовать неуправляемый код (листинг 8.17).

**Листинг 8.17. Состояние клавиш-индикаторов через неуправляемый код**

```
[DllImport("user32")]
public static extern int GetKeyboardState(byte[] pbKeyState);

private void butStatus_Click(object sender, EventArgs e)
{
    bool CapsLockState;
    byte[] keyState = new byte[256];

    GetKeyboardState(keyState);

    CapsLockState = Convert.ToBoolean(keyState[VK_CAPITAL]);

    if (CapsLockState)
        lblCapsLock.Text = "Caps Lock: Вкл";
    else
        lblCapsLock.Text = "Caps Lock: Выкл";
}
```

С помощью данного кода можно было узнать состояние клавиши <Caps Lock>. Но с появлением .NET Framework 2.0 задача упростилась. В классе `Control`

появился новый метод `IsKeyLocked`, который определяет состояние трех клавиш <Caps Lock>, <Num Lock> и <Scroll Lock>. Работа с ним проиллюстрирована в листинге 8.18.

**Листинг 8.18. Определение состояния клавиш-индикаторов с помощью управляемого кода**

```
// Получаем текущее состояние клавиши Caps Lock
lblCapsLock.Text = Control.IsKeyLocked(Keys.CapsLock).ToString();
```

> ***ПРИМЕЧАНИЕ***
>
> Пример, иллюстрирующий работу с клавишами-индикаторами, находится в папке CapsLock на прилагаемом диске.

## Последнее нажатие на клавишу или на кнопку мыши

Наверное, вы замечали, что ICQ и ей подобные программы могут определять активность пользователя. Если пользователь в течение определенного времени не прикасался к клавишам клавиатуры или не трогал мышь, то значок программы меняется на состояние **Отошел**. И его собеседник, видя это состояние, уже не будет понапрасну посылать сообщения. Для реализации этой задачи очень удобно использовать функцию Windows API `GetLastInputInfo`, которая возвращает время, прошедшее с последнего нажатия на клавиатуру или мышь. Разместим на форме кнопку и таймер. Все свойства элементов будут определены программно, поэтому оставляем все значения свойств по умолчанию (листинг 8.19).

**Листинг 8.19. Определение времени с момента последнего нажатия клавиши клавиатуры или движения мышью**

```
struct LASTINPUTINFO
{
    public int cbSize;
    public int dwTime;
}

[DllImport("user32.dll")]
static extern bool GetLastInputInfo(ref LASTINPUTINFO plii);
```

```
static int GetLastInputTime()
{
    int idleTime = 0;
    LASTINPUTINFO lastInputInfo = new LASTINPUTINFO();
    lastInputInfo.cbSize = Marshal.SizeOf(lastInputInfo);
    lastInputInfo.dwTime = 0;

    int envTicks = Environment.TickCount;

    if (GetLastInputInfo(ref lastInputInfo))
    {
        int lastInputTick = lastInputInfo.dwTime;

        idleTime = envTicks - lastInputTick;
    }

    return ((idleTime > 0) ? (idleTime / 1000) : idleTime);
}

private void button1_Click(object sender, EventArgs e)
{
    timer1.Enabled = !timer1.Enabled;

    if (timer1.Enabled == true)
    {
        button1.Text = "Стоп";
    }
    else
    {
        button1.Text = "Старт";
    }
}

private void Form1_Load(object sender, EventArgs e)
{
    button1.Text = "Стоп";
    timer1.Interval = 10;
    timer1.Enabled = true;
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
```

```
    timer1.Enabled = false;
}

private void timer1_Tick(object sender, EventArgs e)
{
    this.Text = GetLastInputTime().ToString();
}
```

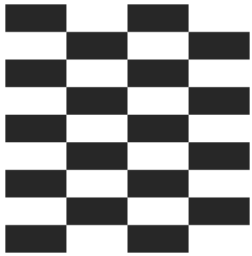
Запустите программу и не предпринимайте никаких действий. Вы увидите в заголовке формы счетчик времени, показывающий бездействие пользователя. Стоит вам пошевелить мышкой или нажать на любую клавишу, как счетчик сбросит свои показания и начнет новый отсчет.

> ***Примечание***
>
> Пример работы с функцией `GetLastInputInfo` находится в папке LastInput.

# Заключение

Пользователь взаимодействует с программой, вводя информацию посредством мыши или клавиатуры. Поэтому разработке удобного интерфейса нужно уделять особое внимание. Существуют различные рекомендации, как грамотно размещать элементы управления и управлять логикой программы через устройства ввода. Отнеситесь к этому аспекту программирования со всей серьезностью.

## Глава 9



# Приложения

В предыдущих главах мы рассмотрели приемы программирования, связанные с графикой, формами, элементами управлениями и устройствами ввода. Эти компоненты являются кирпичиками, из которых складывается полноценная программа. Настало время применить полученные знания на практике и перейти к советам, которые относятся к приложениям.

## Работа с процессами

Часто возникает необходимость узнать, какие процессы запущены на компьютере, и определить параметры запуска, например, путь к приложению, или запустить дочерний процесс.

### Как получить полное имя файла запущенного приложения?

Чтобы получить путь к исполняемому файлу, можно воспользоваться свойством `ExecutablePath` класса `Application`. Его использование проиллюстрировано в листинге 9.1.

**Листинг 9.1. Получение пути к запущенному приложению**

```
private void butExePath_Click(object sender, EventArgs e)
{
    string appPath = Application.ExecutablePath;
    // Выводим полный путь к файлу
    MessageBox.Show(appPath);
}
```

***Примечание***

В .NET Compact Framework свойство ExecutablePath не поддерживается. Вам придется воспользоваться другим способом, описанным в моей книге "Программирование КПК и смартфонов на .NET Compact Framework".

## Как получить путь к папке, из которой запущено приложение?

Не всегда нужно знать путь к исполняемому файлу, иногда требуется получить путь к папке, из которой запущено приложение. Теоретически можно из строки предыдущего примера убрать имя файла и получить, таким образом, путь к папке. Но есть готовое свойство `BaseDirectory`, которым гораздо проще воспользоваться для решения этой задачи. А есть еще свойство `Application.StatupPath`, которое также возвращает путь к папке, но без косой черты в конце строки. Использование этих методов показано в листинге 9.2. Несмотря на сходство, эти свойства имеют некоторые различия, и бывают ситуации, когда можно применять только одно из них.

**Листинг 9.2. Получение пути к папке, из которого запущено приложение**

```
private void butBaseDirectory_Click(object sender, EventArgs e)
{
    // Выводим путь к папке, откуда запущено приложение
    MessageBox.Show(System.AppDomain.CurrentDomain.BaseDirectory);

    // Другой способ
    MessageBox.Show(Application.StartupPath);
}
```

## Как запустить другой исполняемый файл из своего приложения?

Чтобы из своего приложения запустить другую программу, используйте класс `Process` пространства имен `System.Diagnostics` (листинг 9.3).

**Листинг 9.3. Запуск другого приложения**

```
using System.Diagnostics;

// создаем новый процесс
Process proc = new Process();
// Запускаем Блокнот
proc.StartInfo.FileName = @"Notepad.exe";
proc.StartInfo.Arguments = "";
proc.Start();
```

В этом примере мы запускаем пустой текстовый редактор. Если вы хотите запустить Блокнот с открытым файлом, или же любую другую программу с параметрами, то используйте перегруженную версию метода `Start`. Например, для запуска Блокнота с указанным файлом или для запуска Internet Explorer с определенным адресом используйте код, приведенный в листинге 9.4.

**Листинг 9.4. Запуск приложения с параметрами**

```
// Запускаем Блокнот с файлом test.txt
Process.Start("notepad.exe", "test.txt");

// Запускаем браузер с заданным адресом
Process.Start("iexplore.exe", "netsources.narod.ru");
```

Бывает и такая ситуация, когда нужно запустить другое приложение из своей программы и не дать пользователю возможности закрыть нашу программу. В этом случае код немного усложнится (листинг 9.5).

**Листинг 9.5. Запуск приложения с запретом на закрытие своего приложения**

```
private void butRunNotepad2_Click(object sender, EventArgs e)
{
    // Устанавливаем информацию
    ProcessStartInfo start_info = new ProcessStartInfo
        (@"C:\windows\system32\notepad.exe");

    start_info.UseShellExecute = false;
    start_info.CreateNoWindow = true;
    // создаем новый процесс
    Process proc = new Process();
```

```
    proc.StartInfo = start_info;
    // Запускаем процесс
    proc.Start();

    // Ждем, пока Блокнот запущен
    proc.WaitForExit();

    MessageBox.Show("Код завершения: " + proc.ExitCode, "Завершение " +
        "Код", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

## Как закрыть все копии Блокнота?

Впрочем, с помощью класса `Process` можно не только запускать приложения, но и закрывать их. Предположим, мы хотим закрыть все запущенные копии текстового редактора Блокнот. Тогда можно воспользоваться кодом, представленным в листинге 9.6.

**Листинг 9.6. Закрытие всех копий программы Блокнот**

```
using System.Diagnostics;

protected Process[] procs;

private void butCloseNotepad_Click(object sender, EventArgs e)
{
    procs = Process.GetProcessesByName("Notepad");
    int i = 0;
    while (i != procs.Length)
    {
        procs[i].Kill();
        i++;
        MessageBox.Show("Всего : " + i.ToString());
    }
}
```

При этом все экземпляры Блокнота будут закрыты без вывода диалогового окна о необходимости сохранить данные. Поэтому этим способом следует пользоваться осторожно. Естественно, этим приемом можно пользоваться, чтобы закрыть все экземпляры браузера Internet Explorer и другие программы.

## Запуск программы по имени файла

Можно запускать не только исполняемый файл, но приложение, ассоциированное с расширением файла. Например, текстовый файл откроется Блокнотом, а графический рисунок с расширением bmp откроется в программе Paint.

Добавим в начале кода строчку

```
using System.Diagnostics;
```

Теперь необходимо установить данные для структуры `ProcessStartInfo`. Воспользуемся полем структуры, отвечающей за имя файла. Другое поле `UseShellExecute` задает процесс, который запустится на основе расширения файла или типа файла вместо исполняемого файла. Установите это свойство в значение `true`.

```
string sysFolder =
    Environment.GetFolderPath(Environment.SpecialFolder.System);
ProcessStartInfo pInfo = new ProcessStartInfo();
pInfo.FileName = sysFolder + @"\eula.txt";
pInfo.UseShellExecute = true;
Process p  = Process.Start(pInfo);
```

Этот пример открывает файл eula.txt в программе, которая ассоциирована с текстовыми файлами. Как правило, это Блокнот (Notepad.exe). Вы можете изменить имя и тип файла, и он также будет открыт в соответствующем приложении.

Поскольку по умолчанию `UseShellExecute` равно `true`, в этом случае не обязательно использовать `ProcessStartInfo` для запуска процесса. Можно запустить ассоциированное приложение с одной строчкой кода:

```
Process p  = Process.Start(@"C:\winnt\system32\eula.txt");
```

Полный код приведен в листинге 9.7.

**Листинг 9.7. Открытие файла в ассоциированном приложении**

```
using System.Diagnostics;

    // Получим путь к нужной папке
    string sysFolder =
    Environment.GetFolderPath(Environment.SpecialFolder.System);
    // Создадим новую структуру ProcessStartInfo
    ProcessStartInfo pInfo = new ProcessStartInfo();
    // Установим полный путь к имени файла
    pInfo.FileName = sysFolder + @"\eula.txt";
```

```
    // По умолчанию UseShellExecute равно true.
    // В данном случае указываем это поле явно для иллюстрации.
    pInfo.UseShellExecute = true;
    Process p  = Process.Start(pInfo);
```

Возможна ситуация, когда компьютер не имеет программ, ассоциированных с файлом. Поэтому неплохо бы использовать конструкцию `try...catch` для отлавливания подобных ошибок.

## Как узнать число процессоров в системе?

Вы можете узнать число процессоров в системе при помощи свойства `ProcessorAffinity`, как показано в листинге 9.8.

**Листинг 9.8. Получение числа процессоров в системе**

```
using System.Diagnostics;

private void butProcCount_Click(object sender, EventArgs e)
{
    int am = Process.GetCurrentProcess().ProcessorAffinity.ToInt32();

    int processorCount = 0;

    while (am != 0)
    {
        processorCount++;
        am &= (am - 1);
    }

    MessageBox.Show(processorCount.ToString());
}
```

Впрочем, этот код устарел для .NET Framework 2.0, так как число процессоров можно узнать с помощью одной строчки кода (листинг 9.9) с использованием свойства `Environment.ProcessorCount`.

**Листинг 9.9. Второй способ получения процессоров в системе**

```
private void butProcCount2_Click(object sender, EventArgs e)
{
```

```
    MessageBox.Show(
        String.Format(
        "Число процессоров:  {0}",
            Environment.ProcessorCount.ToString()));
}
```

## Как приостановить выполнение программы на несколько секунд?

Если в вашей программе необходимо сделать паузу на несколько секунд, то самым лучшим решением (листинг 9.10) будет воспользоваться методом `Sleep` из класса `System.Threading.Thread`.

**Листинг 9.10. Пауза в программе**

```
// Делаем паузу на 3 секунды
System.Threading.Thread.Sleep(3000);
```

Ни в коем случае не используйте пустые циклы для создания пауз. Этим приемом часто пользовались программисты в играх несколько лет назад, чтобы заморозить выполнение программ. Но с ростом производительности компьютерных систем этот прием принес головную боль пользователям. Мощные процессоры так быстро прогоняли циклы, что играть становилось просто невозможно.

***ПРИМЕЧАНИЕ***

Примеры находятся в папке Application на прилагаемом диске.

## Как получить список всех процессов, запущенных в системе?

Класс `Process` из пространства имен `System.Diagnostics` позволяет узнать множество информации о процессах. Например, с помощью метода `Process.GetProcesses` не составит труда получить список всех запущенных процессов на локальной машине, как показано в листинге 9.11.

**Листинг 9.11. Получение списка всех процессов, запущенных в системе**

```
using System.Diagnostics;

private void Form1_Load(object sender, EventArgs e)
{
    foreach (Process p in Process.GetProcesses())
       listBox1.Items.Add(p.ToString());
}
```

## Как получить список только оконных процессов на моей машине?

Можно получить список только оконных процессов, если проверять свойство `MainWindowHandle` (листинг 9.12). Мы также передаем методу `GetProcesses` имя компьютера, в этом случае равное значению свойства `System.Environment.MachineName`. Результат показан на рис. 9.1.

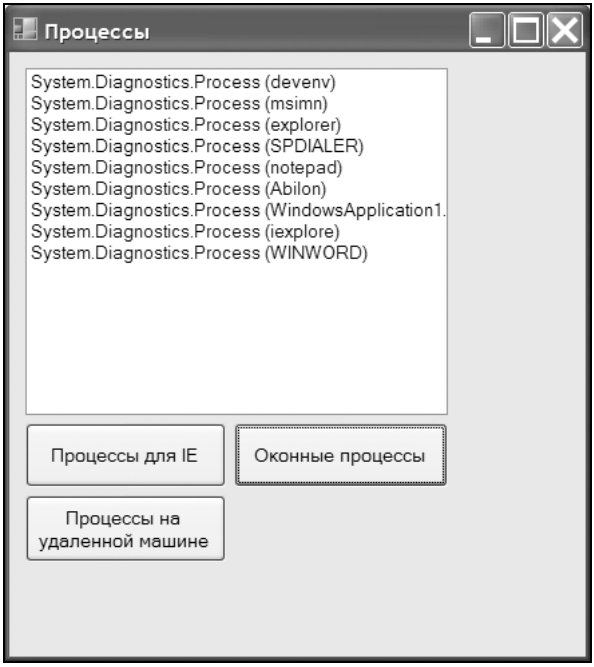



**Рис. 9.1.** Получение списка оконных процессов

**Листинг 9.12. Получение списка оконных процессов**

```
private void butWindowProcess_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();

    foreach (Process p in
                Process.GetProcesses(System.Environment.MachineName))
    {
        if (p.MainWindowHandle != IntPtr.Zero)
        {
            // Оконные приложения
            listBox1.Items.Add(p.ToString());
        }
    }
}
```

## Как получить список определенных процессов?

Можно получить список определенных процессов. Например, в листинге 9.13 мы получаем процессы, связанные с браузером Internet Explorer.

**Листинг 9.13. Получение списка определенных процессов**

```
using System.Diagnostics;

private void butProcessIE_Click(object sender, EventArgs e)
{
    // Очистим список
    listBox1.Items.Clear();
    // Получим список процессов, связанных с Internet Explorer
    foreach (Process p in Process.GetProcessesByName("iexplore"))
        listBox1.Items.Add(p.ToString());
}
```

## Получение списка процессов на удаленной машине

Можно также получить информацию об определенных процессах и на удаленной машине, используя перегруженную версию метода `GetProcessesByName`, в котором надо указать имя удаленной машины без обратных двойных косых

черт (\\). Например, в следующем примере (листинг 9.14) мы пытаемся получить список Блокнотов, запущенных на удаленной машине skynet.

**Листинг 9.14. Получение процессов на удаленной машине**

```
using System.Diagnostics;

private void butRemoteProcess_Click(object sender, EventArgs e)
{
    // Очистим список
    listBox1.Items.Clear();
    // Получим список процессов notepad на удаленной машине skynet
    foreach (Process p in Process.GetProcessesByName("notepad",
                                                     "skynet"))
        listBox1.Items.Add(p.ToString());
}
```

## Как открыть почтовый клиент, установленный по умолчанию, и установить необходимые параметры для отправки письма?

Продолжим опыты с классом `Process`. Предположим, мы хотим предусмотреть в нашей программе возможность отправки письма автору программы, в котором пользователь может высказать все, что он думает об авторе и его великолепном (или ужасном) творении. Для начала надо сформировать необходимые атрибуты для отправки письма (листинг 9.15). Далее нужно использовать команду `MailTo` для запуска почтового клиента с подготовленным письмом. Минимальный синтаксис такой команды должен выглядеть следующим образом: `MailTo:email@address1.com`. Кроме того, можно задавать дополнительные параметры, перечисленные в табл. 9.1.

***Таблица 9.1.*** *Правила формирования письма*

| Возможность | Синтаксис |
|---|---|
| Несколько получателей письма | `email1@site.com,email2@site2.com` (адреса разделяются запятыми) |
| Добавление текста в тему письма (Subject) | `?subject=Ваш текст для темы письма` |

***Таблица 9.1*** *(окончание)*

| Возможность | Синтаксис |
|---|---|
| Добавление адреса в поле **Копия** (Copy To или CC) | `&cc=bg@microsoft.com` |
| Добавение адреса в поле **Скрытая** (Blind Copy To или BCC) | `&bcc=alla@pugacheva.ru` |
| Задание содержимого письма (Body) | `&body=Здесь текст вашего письма` |

После того как мы сформировали команду для отправки письма, можно его отправлять через почтовый клиент, установленный у пользователя по умолчанию. Когда вы запустите проект и нажмете на кнопку **Послать письмо**, то у вас откроется ваш почтовый клиент, в котором уже сформировано письмо. Примером такого клиента могут быть такие программы, как Outlook, Outlook Express, The Bat!, Eudora и др. После этого остается подправить текст, сформированный программно, и отправить письмо средствами установленного у вас по умолчанию клиента (листинг 9.15).

**Листинг 9.15. Отправка письма через почтовый клиент, установленный по умолчанию**

```
using System.Diagnostics.Process;

Process process = new Process();
process.StartInfo.FileName =
    "mailto:email@address1.com,email@address2.com?subject=Hello
&cc=email@address3.com&bcc=email@address4.com&body=Happy New Year";
process.Start();
```

***Примечание***

Примеры находятся в папке ProcessesSamples на прилагаемом диске.

# Определение операционной системы пользователя

Для определения операционной системы нужно использовать свойство `OSVersion` класса `Environment` из пространства имен `System`. Свойство

`Version` имеет поля `Major` и `Minor`, которые содержат дополнительную информацию. Операционная система Windows XP имеет версию 5.1 и сборку 2600, а новая Windows Vista имеет версию 6.0 и сборку 6600. В статье "How To Determine the Operating System Service Pack Level in Visual C# .NET" **(http://support.microsoft.com/kb/304721/EN-US/)** приводится пример получения номера сервис-пака с помощью функции Windows API `GetVersionEx`. Данный пример был предназначен для использования в .NET Framework 1.1, но с появлением .NET Framework 2.0 надобность в использовании неуправляемого кода отпала. В составе новой платформы .NET Framework 2.0 появились такие новые свойства, как `ServicePack` и `VersionString` класса `OperatingSystem`, которые позволяют быстро получить необходимые данные (листинг 9.16).

**Листинг 9.16. Определение операционной системы пользователя**

```
OperatingSystem os = Environment.OSVersion;
listBox1.Items.Add(os.Version);
listBox1.Items.Add(os.Platform);
listBox1.Items.Add(os.ServicePack);
listBox1.Items.Add(os.VersionString);
```

Особое внимание нужно обратить тем программистам, которые писали программы специально для Windows XP. После выхода этой операционной системы многие разработчики в своих проектах делали проверку на наличие этой системы у пользователя, чтобы задействовать предоставляемые ей новые возможности. Проверка могла осуществляться такой строчкой:

```
if ((os.Version.Major == 5) && (os.Version.Minor == 1)) // и так далее
```

В данном примере проверяется наличие операционной системы Windows XP. Если вы запустите проект с подобной проверкой под Windows Vista, то программа будет считать, что Windows Vista не входит в число поддерживаемых операционных систем. Поэтому вам нужно переделать эту проверку, как показано в листинге 9.17.

**Листинг 9.17. Исправленная версия проверки операционной системы**

```
private void CheckOSVersion()
{
    OperatingSystem os = Environment.OSVersion;
    Version version = os.Version;
    if ((version.Major == 5) && (version.Minor == 1)
              || version.Major >= 6)
```

```
    {
        MessageBox.Show("Программа может запускаться" +
                        " в вашей операционной системе");
    }
    else
    {
        MessageBox.Show
            ("Эта версия операционной системы не поддерживается." +
            "\r\n Используйте Windows XP или Windows Vista");
     }
}

private void butCheckOS_Click(object sender, EventArgs e)
{
    CheckOSVersion();
}
```

***Примечание***

Пример находится в папке OperatingSystemDemo на прилагаемом диске.

# Определение версии .NET Framework и ее сервис-пака

С тех пор, как Microsoft представила публике первую версию .NET Framework, утекло много воды. Были выпущены последовательно версии 1.0, 1.1, 2.0, 3.0 и 3.5, и перед разработчиками встала проблема определения установленных версий .NET Framework на пользовательской машине. Архитектура .NET позволяет сосуществовать нескольким версиям .NET Framework одновременно. Причем приложение, созданное, к примеру, при помощи .NET Framework 1.0, может использовать новую версию .NET Framework или попрежнему использовать ту версию платформы, в которой оно было создано (по умолчанию). Определение версии .NET Framework, как правило, важно при установке программы на компьютер, а также при использовании каких-то новых методов, свойств, классов, не поддерживаемых в старых версиях. Самый простой и верный способ определения версии .NET Framework — считать информацию из определенной ветки реестра. В каждом новом релизе .NET Framework в реестре вносятся определенные изменения. Рассмотрим этот вопрос подробнее.

## .NET Framework 1.0

При установке .NET Framework 1.0 в реестре появляется строковый параметр 4322 в разделе:

**HKLM\Software\Microsoft\.NETFramework\Policy\v1.0**

Параметр 4322 определяет номер сборки и может различаться на разных компьютерах.

## .NET Framework версий 1.1, 2.0 3.0 и v3.5 (Orcas)

Специалисты Microsoft сочли, что такая реализация метода кодирования версии не слишком удачна, и, начиная с .NET Framework 1.1, стали применять другой подход для кодирования версий. Теперь при установке новой версии .NET Framework вносятся записи в раздел реестра

**HKLM\Software\Microsoft\NET Framework Setup\NDP**

Записи в реестре, соответствующие различным версиям .NET Framework, делаются в разделах, перечисленных в табл. 9.2.

***Таблица 9.2.*** *Записи в реестре*

| Версия | Раздел реестра |
|---|---|
| .NET Framework 1.1 | **HKLM\Software\Microsoft\NET Framework Setup\NDP\v1.1.4322** |
| .NET Framework 2.0 | **HKLM\Software\Microsoft\NET Framework Setup\NDP\v2.0.50727** |
| .NET Framework 3.0 | **HKLM\Software\Microsoft\NET Framework Setup\NDP\v3.0** |
| .NET Framework 3.5 | **HKLM\Software\Microsoft\NET Framework Setup\NDP\v3.5** |

В этих разделах (за исключением .NET Framework 3.0) имеется параметр типа DWORD **Install**, который имеет значение 1, если соответствующая версия .NET Framework установлена. Также в этих разделах (кроме .NET Framework 3.0) имеется еще один параметр типа DWORD **SP**, который показывает нали-

чие установленного сервис-пака. Для .NET Framework 2.0 и 3.5 в этом же разделе реестра имеются еще параметры типа DWORD **Increment** (для 2.0) и **Version** (3.5), которые содержат номер сборки (build).

## .NET Framework 3.0

Как вы обратили внимание, версия .NET Framework 3.0 стоит немного особняком. Данная версия использует несколько собственных записей в реестре, связанных с новыми библиотеками Foundation, которые вошли в состав платформы. Для определения версии платформы и установленных библиотек используются записи в реестре, перечисленные в табл. 9.3.

***Таблица 9.3.*** *Настройки в реестре для .NET Framework 3.0*

| Версия .NET Framework | Ключ реестра |
|---|---|
| Windows Communication Foundation | **HKLM\Software\Microsoft\NET Framework Setup\NDP\v3.0\Setup\Windows Communication Foundation** |
| Windows Presentation Foundation | **HKLM\Software\Microsoft\NET Framework Setup\NDP\v3.0\Setup\Windows Presentation Foundation** |
| Windows Workflow Foundation | **HKLM\Software\Microsoft\NET Framework Setup\NDP\v3.0\Setup\Windows Workflow Foundation** |

В указанных в таблице разделах имеется параметр типа DWORD **InstallSucces**, который свидетельствует об успешной установке соответствующей библиотеки. Здесь также может присутствовать параметр **SP**, который свидетельствует об установленном сервис-паке. А строковый параметр **Version** или **FileVersion** (для Windows Workflow Foundation) указывает на точный номер версии .NET Framework. За более детальной информацией обратитесь к статье "Using managed code to detect what .NET Framework versions and service packs are installed", которую можно найти по адресу **http://www.codeproject.com/useritems/frameworkversiondetection.asp**. Автор статьи пошел гораздо дальше меня в изысканиях по этому вопросу и написал всеобъемлющий код, позволяющий извлечь подробную информацию обо всех версиях и сервис-

паках .NET Framework, установленных на компьютере. Так как код оригинальной программы слишком велик, я написал свой укороченный вариант этой программы (листинг 9.18), который позволяет извлечь информацию только о двух версиях — 1.1 и 2.0.

**Листинг 9.18. Получение списка версий .NET Framework**

```
// При написании программы использовался исходный код
// из примера
// Using managed code to detect what .NET Framework versions
// and service packs are installed
// который можно найти по адресу
// http://www.codeproject.com/useritems/frameworkversiondetection.asp

using Microsoft.Win32;
using System.Globalization;

const string Netfx11RegKeyName =
    "Software\\Microsoft\\NET Framework Setup\\NDP\\v1.1.4322";
const string Netfx20RegKeyName =
    "Software\\Microsoft\\NET Framework Setup\\NDP\\v2.0.50727";
const string Netfx11PlusRegValueName = "Install";
const string Netfx11PlusSPxRegValueName = "SP";
const string Netfx20PlusBuildRegValueName = "Increment";

/// <summary>
/// Версии .NET Framework
/// </summary>
public enum FrameworkVersion
{
    // .NET Framework 1.0
    Fx10,
    // .NET Framework 1.1
    Fx11,
    // .NET Framework 2.0
    Fx20,
    // .NET Framework 3.0
    Fx30,
    // .NET Framework 3.5 (Orcas)
    Fx35,
}
```

```
private void button1_Click(object sender, EventArgs e)
{
    bool fx11Installed = IsInstalled(FrameworkVersion.Fx11);
    bool fx20Installed = IsInstalled(FrameworkVersion.Fx20);

    listBox1.Items.Add(String.Format(".NET Framework 1.1 installed? {0}",
                                     fx11Installed));

    if (fx11Installed)
    {
        listBox1.Items.Add(String.Format(
            ".NET Framework 1.1 Exact Version: {0}",
                GetExactVersion(FrameworkVersion.Fx11)));
        listBox1.Items.Add(String.Format(
            ".NET Framework 1.1 Service Pack: {0}",
                GetServicePackLevel(FrameworkVersion.Fx11)));
    }

    listBox1.Items.Add(String.Format(
        ".NET Framework 2.0 installed? {0}", fx20Installed));
    if (fx20Installed)
    {
        listBox1.Items.Add(String.Format(
            ".NET Framework 2.0 Exact Version: {0}",
                GetExactVersion(FrameworkVersion.Fx20)));
        listBox1.Items.Add(String.Format(
            ".NET Framework 2.0 Service Pack: {0}",
                GetServicePackLevel(FrameworkVersion.Fx20)));
    }
}

/// <summary>
/// Определяет, установлена ли указанная версия .NET Framework
/// на локальном компьютере.
/// </summary>
/// <param name="frameworkVersion">Одно из значений
/// <see cref="FrameworkVersion"/>.</param>
/// <returns><see langword="true"/> если указанная версия .NET Framework
/// установлена; иначе <see langword="false"/>.</returns>
public static bool IsInstalled(FrameworkVersion frameworkVersion)
{
    bool ret = false;
```

```
    switch (frameworkVersion)
    {
        case FrameworkVersion.Fx11:
            ret = IsNetfx11Installed();
            break;

        case FrameworkVersion.Fx20:
            ret = IsNetfx20Installed();
            break;

        default:
            break;
    }

    return ret;
}

private static bool IsNetfx11Installed()
{
    bool found = false;
    int regValue = 0;

    if (GetRegistryValue(RegistryHive.LocalMachine,
        Netfx11RegKeyName, Netfx11PlusRegValueName,
        RegistryValueKind.DWord, out regValue))
    {
        if (regValue == 1)
        {
            found = true;
        }
    }

    return found;
}

private static bool IsNetfx20Installed()
{
    bool found = false;
    int regValue = 0;

    if (GetRegistryValue(RegistryHive.LocalMachine,
        Netfx20RegKeyName, Netfx11PlusRegValueName,
```

```
        RegistryValueKind.DWord, out regValue))
    {
        if (regValue == 1)
        {
            found = true;
        }
    }

    return found;
}

private static bool GetRegistryValue<T>(RegistryHive hive, string key,
    string value, RegistryValueKind kind, out T data)
{
    bool success = false;
    data = default(T);

    using (RegistryKey baseKey =
               RegistryKey.OpenRemoteBaseKey(hive, String.Empty))
    {
        if (baseKey != null)
        {
            using (RegistryKey registryKey = baseKey.OpenSubKey(key,
                RegistryKeyPermissionCheck.ReadSubTree))
            {
                if (registryKey != null)
                {
                    // Если ключ был открыт, попытаться извлечь значение.
                    RegistryValueKind kindFound =
                        registryKey.GetValueKind(value);
                    if (kindFound == kind)
                    {
                        object regValue =
                            registryKey.GetValue(value, null);
                        if (regValue != null)
                        {
                            data = (T)Convert.ChangeType(regValue,
                                typeof(T),
                                CultureInfo.InvariantCulture);
                            success = true;
                        }
                    }
```

```
                }
            }
        }
    }
    return success;
}

/// <summary>
/// Возвращает точный номер версии для указанной версии .NET Framework.
/// </summary>
/// <param name="frameworkVersion">Одно из значений
/// <see cref="FrameworkVersion"/>.</param>
/// <returns>Значение <see cref="Version">version</see>, представляющее
/// точный номер версии для указанной версии .NET Framework.
/// Если указанная версия .NET Framework не найдена, возвращается
/// значение <see cref="Version"/>, которое представляет
/// номер версии 0.0.0.0.
/// </returns>
public static Version GetExactVersion(FrameworkVersion frameworkVersion)
{
    Version fxVersion = new Version();

    switch (frameworkVersion)
    {
        case FrameworkVersion.Fx11:
            fxVersion = GetNetfx11ExactVersion();
            break;

        case FrameworkVersion.Fx20:
            fxVersion = GetNetfx20ExactVersion();
            break;

        default:
            break;
    }

    return fxVersion;
}

private static Version GetNetfx11ExactVersion()
{
    int regValue = 0;
```

```
    // Мы можем получить -1 только, если .NET Framework
    // не установлена или произошла какая-то ошибка при извлечении
    // данных из реестра
    Version fxVersion = new Version();

    if (GetRegistryValue(RegistryHive.LocalMachine,
        Netfx11RegKeyName, Netfx11PlusRegValueName,
        RegistryValueKind.DWord, out regValue))
    {
        if (regValue == 1)
        {
            // В строгом смысле мы здесь мухлюем, но имя ключа реестра
            // само содержит номер версии.
            string[] tokens = Netfx11RegKeyName.Split(
                new string[] { "NDP\\v" }, StringSplitOptions.None);
            if (tokens.Length == 2)
            {
                fxVersion = new Version(tokens[1]);
            }
        }
    }

    return fxVersion;
}

private static Version GetNetfx20ExactVersion()
{
    string regValue = String.Empty;

    // Мы можем получить -1 только, если .NET Framework
    // не установлена или произошла какая-то ошибка при извлечении
    // данных из реестра
    Version fxVersion = new Version();

    if (GetRegistryValue(RegistryHive.LocalMachine,
        Netfx20RegKeyName, Netfx20PlusBuildRegValueName,
        RegistryValueKind.String, out regValue))
    {
        if (!String.IsNullOrEmpty(regValue))
        {
            // В строгом смысле мы здесь мухлюем, но имя ключа реестра
            // само содержит номер версии.
```

```
            string[] versionTokens = Netfx20RegKeyName.Split(
                new string[] { "NDP\\v" }, StringSplitOptions.None);
            if (versionTokens.Length == 2)
            {
                string[] tokens = versionTokens[1].Split('.');
                if (tokens.Length == 3)
                {
                    fxVersion = new Version(
                        Convert.ToInt32(tokens[0],
                            NumberFormatInfo.InvariantInfo),
                        Convert.ToInt32(tokens[1],
                            NumberFormatInfo.InvariantInfo),
                        Convert.ToInt32(tokens[2],
                            NumberFormatInfo.InvariantInfo),
                        Convert.ToInt32(regValue,
                            NumberFormatInfo.InvariantInfo));
                }
            }
        }
    }

    return fxVersion;
}

/// <summary>
/// Возвращает сервис-пак для указанной версии .NET Framework.
/// </summary>
/// <param name="frameworkVersion">Одно из значений
/// <see cref="FrameworkVersion"/>.</param>
/// <returns>Значение <see cref="Int32">integer</see>, представляющее
/// собой сервис-пак для указанной версии .NET Framework. Если
/// указанная версия .NET Framework не найдена, возвращается -1.
/// </returns>
public static int GetServicePackLevel(FrameworkVersion frameworkVersion)
{
    int servicePackLevel = -1;

    switch (frameworkVersion)
    {
        case FrameworkVersion.Fx11:
            servicePackLevel = GetNetfx11SPLevel();
            break;
```

```
        case FrameworkVersion.Fx20:
            servicePackLevel = GetNetfx20SPLevel();
            break;

        default:
            break;
    }

    return servicePackLevel;
}

private static int GetNetfx11SPLevel()
{
    int regValue = 0;

    // Мы можем получить -1, только если .NET Framework
    // не установлена или произошла какая-то ошибка при извлечении
    // данных из реестра
    int servicePackLevel = -1;

    if (GetRegistryValue(RegistryHive.LocalMachine,
        Netfx11RegKeyName, Netfx11PlusSPxRegValueName,
        RegistryValueKind.DWord, out regValue))
    {
        servicePackLevel = regValue;
    }

    return servicePackLevel;
}

private static int GetNetfx20SPLevel()
{
    int regValue = 0;

    // Мы можем получить -1, только если .NET Framework
    // не установлена или произошла какая-то ошибка при извлечении
    // данных из реестра
    int servicePackLevel = -1;

    if (GetRegistryValue(RegistryHive.LocalMachine,
        Netfx20RegKeyName, Netfx11PlusSPxRegValueName,
        RegistryValueKind.DWord, out regValue))
```

```
    {
        servicePackLevel = regValue;
    }

    return servicePackLevel;
}
```

***Примечание***

Пример находится в папке NetFrameworkVersions на прилагаемом диске.

# Определение папки установки .NET Framework

Иногда требуется узнать, где находится папка установки .NET Framework. В этом случае вам поможет функция `GetCORSystemDirectory`, которая позволяет получить каталог установки той версии библиотеки Common Language Runtime (CLR), которая загружена в текущий процесс. Если CLR не загружена, то функция возвращает каталог, куда установлена самая последняя версия (листинг 9.19).

**Листинг 9.19. Получение имени папки, в которой установлена .NET Framework**

```
using System;
using System.Runtime.InteropServices;

class MainClass
{
    [DllImport("mscoree.dll")]
    private static extern void GetCORSystemDirectory (
    [MarshalAs(UnmanagedType.LPTStr)]
    System.Text.StringBuilder Buffer,
    int BufferLength, ref int Length);

    [STAThread]
    static void Main()
    {
        System.Text.StringBuilder sb =
            new System.Text.StringBuilder(260);
        int size = 0;
```

```
        GetCORSystemDirectory(sb, sb.Capacity, ref size);

        // Выводим на экран что-то типа
        // "C:\WINDOWS\Microsoft .NET\Framework\v2.0.50727\"
        Console.WriteLine(sb);
        Console.ReadLine();
    }
}
```

***ПРИМЕЧАНИЕ***

Пример находится в папке FrameworkFolder на прилагаемом диске.

# Номер сборки

Каждая сборка имеет номер версии, состоящий из четырех частей, которые определены как атрибуты сборки (assembly attribute) в исходном коде проекта. Например, для стандартного проекта Visual C# 2008 эти атрибуты хранятся в файле AssemblyInfo.cs. Если вы откроете этот файл, то найдете там строку, определяющую номер версии сборки:

```
<Assembly: AssemblyVersion("1.0.0.0")>
```

Вы можете изменить номер версии самостоятельно в этом файле. Также вы можете установить номер версии другим способом. Выберите в меню команды **Project | <ИмяПроекта> Properties**, чтобы открыть окно настроек. Выберите в нем вкладку **Application** и щелкните на кнопке **Assembly Information…**. Установите нужный номер версии в текстовых полях, помеченных как **Assembly Version**.

## Обновление номера версии сборки в автоматическом режиме

Если вы часто обновляете свой проект, то изменение номера версии сборки вручную вам очень быстро надоест. Вы можете переложить эту работу на Visual Studio 2005. Для этого в файле AssemblyInfo.cs используйте звездочку (*) для нужного компонента, определяющего номер версии.

```
<Assembly: AssemblyVersion("1.2.*")>
```

или

```
<Assembly: AssemblyVersion("1.2.3.*")>
```

Для генерации номера редакции (revision) используется число секунд, прошедших с полуночи текущего дня, разделенное на 2. Это значение начинается с 0 каждую полночь.

# Вызов файла справки CHM

В пространстве имен `System.Windows.Forms` имеется класс `Help`, который позволяет выводить справку, подготовленную при помощи утилиты HTMLHelp Workshop (файл CHM). Как это сделать, показано в листинге 9.20.

**Листинг 9.20. Вызов файла справки**

```
// Вызываем файл справки
Help.ShowHelp(this, @"d:\help\dotnet.chm","win32map.html");
// Вызываем файл справки и открываем окно указателя
Help.ShowHelpIndex(this, @"d:\help\dotnet.chm");
```

# Получение номера версии файла и другую информацию

Если в Проводнике выбрать какой-нибудь исполняемый файл, щелкнуть на нем правой кнопкой и выбрать из контекстного меню пункт **Свойства**, то вы увидите детальную информацию о файле. Вы можете получить такую же информацию при помощи свойств класса `System.Diagnostics.FileVersionInfo`. Кроме того, данный класс содержит еще множество других свойств. В листинге 9.21 приводится пример получения части информации о файле regedit.exe, который является стандартным редактором реестра в Windows (рис. 9.2).

**Листинг 9.21. Получение информации об исполняемом файле**

```
private void button1_Click(object sender, EventArgs e)
{
    string fileName = @"c:\windows\regedit.exe";
    // Получим информацию о свойствах файла.
    FileVersionInfo fileInfo = FileVersionInfo.GetVersionInfo(fileName);

    listBox1.Items.Add("Комментарии: " + fileInfo.Comments);
    listBox1.Items.Add("Производитель: " + fileInfo.CompanyName);
```

```
    listBox1.Items.Add("Имя файла: " + fileInfo.FileName);
    listBox1.Items.Add("Номер сборки файла: " + fileInfo.FileBuildPart);
    listBox1.Items.Add("Описание файла: " + fileInfo.FileDescription);
    listBox1.Items.Add("Номер версии файла: " + fileInfo.FileVersion);
    listBox1.Items.Add("Основная часть номера версии: " +
        fileInfo.FileMajorPart);
    listBox1.Items.Add("Вспомогательная часть номера версии: " +
        fileInfo.FileMinorPart);
    listBox1.Items.Add("Номер закрытой части файла: " +
        fileInfo.FilePrivatePart);
    listBox1.Items.Add("Авторские права: " + fileInfo.LegalCopyright);
    listBox1.Items.Add("Товарные знаки: " + fileInfo.LegalTrademarks);
    listBox1.Items.Add("Название продукта: " + fileInfo.ProductName);
}
```

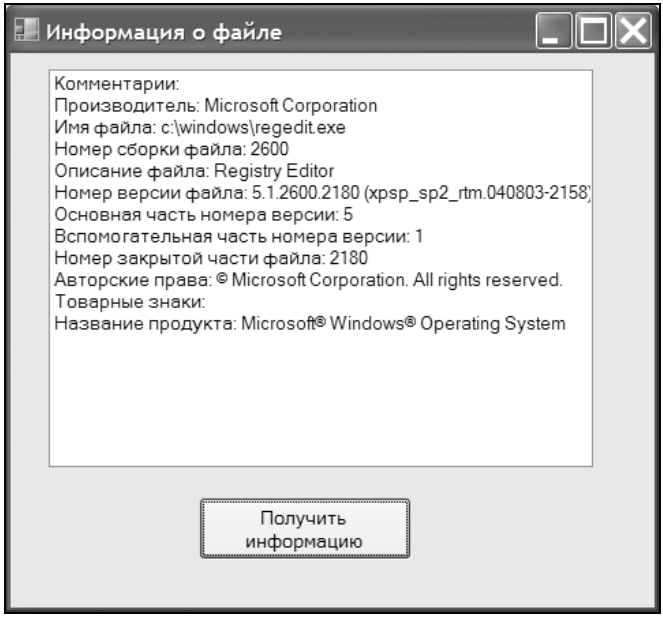



**Рис. 9.2.** Получение информации о файле

### *Примечание*

Пример получения информации о файле находится в папке FileVersionInfoDemo на прилагаемом диске.

# Определение имени пользователя системы

Свойство `SystemInformation.UserName` позволяет получить имя пользователя текущего потока (имя, под которым пользователь в данный момент зарегистрирован в операционной системе). Пример показан в листинге 9.22.

**Листинг 9.22. Получение имени пользователя системы**

```
// Определяем имя пользователя системы
MessageBox.Show(SystemInformation.UserName);
```

Если вы вдруг получили сообщение об ошибке при выполнении программы, то не пугайтесь. Microsoft выпустила по этому поводу специальную статью Базы Знаний "You receive a “System.Security.SecurityException” exception when you run an application that calls the SystemInformation.UserName property", которая доступна по адресу **http://support.microsoft.com/kb/814741/en-us**. Для обхода этой проблемы в статье предлагается воспользоваться схожим свойством `System.Environment.UserName`:

```
MessageBox.Show(System.Enironment.UserName);
```

# Как определить, имеет ли ваша система мышь, узнать число кнопок у мыши, размер вашего монитора и другую информацию?

На самом деле класс `System.Windows.Forms.SystemInformation`, который мы использовали в предыдущем примере, имеет множество других полезных методов и свойств. Например, мы можем узнать, используется ли в системе мышь, сколько у нее кнопок, есть ли колесико, менялись ли местами кнопки у мыши и многое другое. В листинге 9.23 приведен небольшой кусочек кода, который извлекает полезную системную информацию.

**Листинг 9.23. Получение системной информации**

```
private void Form1_Load(object sender, EventArgs e)
{
```

```
    // Получим системную информацию для локальной машины.
    AddSysInfoItem("ComputerName",
        SystemInformation.ComputerName.ToString());
    AddSysInfoItem("UserDomainName",
        SystemInformation.UserDomainName.ToString());
    AddSysInfoItem("UserName", SystemInformation.UserName.ToString());
    AddSysInfoItem("Network", SystemInformation.Network.ToString());
    AddSysInfoItem("Secure", SystemInformation.Secure.ToString());
    AddSysInfoItem("Monitor Count",
        SystemInformation.MonitorCount.ToString());
    AddSysInfoItem("MousePresent",
        SystemInformation.MousePresent.ToString());
    AddSysInfoItem("MouseButtons",
        SystemInformation.MouseButtons.ToString());
    AddSysInfoItem("MouseButtonsSwapped",
        SystemInformation.MouseButtonsSwapped.ToString());
    AddSysInfoItem("Mouse Wheel",
        SystemInformation.MouseWheelPresent.ToString());
    AddSysInfoItem("UserInteractive",
        SystemInformation.UserInteractive.ToString());
}

/// <summary>
/// Получаем пару строковых значений
/// </summary>
/// <param name="property">Свойство</param>
/// <param name="propertyvalue">Значение свойства</param>
private void AddSysInfoItem(string property, string propertyvalue)
{
    ListViewItem lvt;
    lvt = listView1.Items.Add(property);
    lvt.SubItems.Add(propertyvalue);
}
```

### *Примечание*

Пример получения системной информации находится в папке SystemInformationDemo на прилагаемом диске.

# Как зарегистрировать файлы DLL и OCX?

В некоторых случаях программисту приходится регистрировать файлы DLL и OCX, необходимые для работы приложения. Обычно опытные пользователи проводят регистрацию через командную строку с использованием утилиты regsvr32.exe, которая входит в состав операционной системы. Но если возникнет необходимость зарегистрировать файл программным способом, воспользуйтесь примером, приведенным в листинге 9.24.

**Листинг 9.24. Регистрация файлов DLL и OCX**

```
using System.Diagnostics;

// Создадим метод для создания нашего нового процесса
private void createproc(string fname, string arg)
{
    Process proc = new Process();
    proc.StartInfo.FileName = fname;
    proc.StartInfo.Arguments = arg;
    proc.Start();
}

// Теперь с помощью этого метода будем регистрировать dll или ocx.
// fname - имя программы для регистрации файлов.
// arg - задает аргументы для этого файла.
createproc("RegSvr32", " /s \"C:\\WINDOWS\\system32\\shdocvw.dll\"");
// Пример выше регистрирует файл shdocvw.dll без вывода сообщения.
// Мы использовали полный путь файла, но можно чуть изменить пример,
// если файл находится в одной папке с приложением
createproc("RegSvr32", " /s \"" + Application.StartupPath +
                       "\\myappdll.dll\"");
// вот другие параметры, используемые regsvr32:
// optional values []
// regsvr32 [/u] [/s] [/n] [/i[:cmdline]  ] dllname
// /u = снять регистрацию файла
// /s = silent (без вывода сообщения)
// /i = вызвать dllinstall, передав ему необязательный параметр [cmdline]
//      (когда используется с /u, вызывает dll uninstall)
// /n = не вызывать dllregisterserver (опция должна использоваться с /i)
```

***Примечание***

Пример находится в папке RegisterDLL на прилагаемом диске.

# Извлечение строки или значка из ресурсов

Однажды, читая блог Андрея Бороздина, я наткнулся на статью о том, как внедрить строку в ресурс программы, а затем прочитать эту строку программным способом (**http://andybor.blogspot.com/2007/10/blog-post_04.html**). Автор долго и подробно описывал свой способ. А в конце статьи один из читателей оставил свой комментарий, сообщив, что в Visual Studio 2005 эта задача решается намного проще. Автор блога признался, что не знал нового способа и пользовался по старинке старым способом, который использовался в Visual Studio .NET 2003. Этот пример лишний раз показывает, как много изменений произошло в новой версии интегрированной среды разработки. Программисты порой даже и не подозревают об этих изменениях и используют устаревшие способы. Честно говоря, я не удивлюсь, если и на страницах этой книги читатели найдут аналогичные устаревшие приемы программирования, которые можно заменить более прогрессивными вариантами. Но вернемся к ресурсам. Чтобы добавить строку или значок как ресурс в программу и затем извлечь их оттуда, нужно проделать следующее. Выберите в меню **Project** команду **<ИмяПрограммы> Properties** и в открывшемся окне выберите вкладку **Resources**. В этом окне вы можете добавить свои строки и значки. После того как вы проделали эти несложные операции, вы можете извлечь данные из ресурсов, как показано в листинге 9.25.

**Листинг 9.25. Извлечение данных из ресурсов**

```
private void button1_Click(object sender, EventArgs e)
{
    // Извлекаем строку из ресурса String1
    MessageBox.Show(WindowsApplication1.Properties.Resources.String1);

    // Извлекаем значок из ресурса Icon1
    // и устанавливаем его в качестве значка формы
    this.Icon = WindowsApplication1.Properties.Resources.Icon1;
}
```

# Сохранение настроек приложения

.NET Framework 2.0 позволяет очень легко создавать, сохранять и загружать сохраненные настройки работы приложения. Очень часто возникает потребность, чтобы программа запоминала размер шрифта, цвет фона формы и другие настройки, сделанные пользователем, чтобы при следующем запуске приложения она восстановила эти настройки. Рассмотрим, как реализован этот механизм в Visual Studio 2005. Настройки бывают двух видов: пользовательские настройки и настройки приложения. Сами настройки имеют четыре свойства — **Name** (Имя), **Type** (Тип), **Scope** (Область видимости) и **Value** (Значение):

- свойство **Name** определяет имя настройки, по которому можно обратиться к нужной настройке из программы для получения значения во время работы;
- свойство **Type** определяет тип настройки, например, это может быть цвет (`System.Color`), строка (`string`), число и т. д.;
- свойство **Scope** определяет степень доступа к настройке во время выполнения программы. Данное свойство имеет два возможных значения: **Application** и **User**. Настройка на уровне приложения является более важной и влияет на работу программы в целом, а настройка на уровне пользователя больше подходит, например, для удобного оформления цветовых схем и не влияет на работу приложения в целом;
- свойство **Value** определяет значение, возвращаемое при обращении к настройке. Значение может иметь тип, определенный свойством **Type**.

Обратите внимание, что пользовательские настройки можно менять во время работы программы, то есть их можно считывать и записывать (read/write). Программные настройки доступны только для чтения. Изменять программные настройки можно во время разработки программы или путем изменения файла настроек вручную. Рассмотрим пример создания новой настройки во время разработки приложения. Создайте новый проект и выберите в меню **Project | <ИмяПроекта> Properties…** В открывшемся диалоговом окне выберите с левой стороны вкладку **Settings**. У вас откроется окно дизайнера настроек (рис. 9.3).

Щелкните два раза на первой строчке со словом **Settings**, чтобы изменить имя настройки. Введите новое имя, например, **BackgroundColor**. Далее вам надо сопоставить этому имени тип настройки. Выберите из списка подходящий вам тип. В нашем случае это будет `System.Color`. Теперь мы задаем

уровень доступа. Остановимся на значении **User**. Переходим к столбцу **Value**. Так как мы определили тип `Color`, то в Visual Studio для выбора значения предлагается воспользоваться диалоговым окном выбора цвета. Выбираем любой цвет, например **Cyan**. Первая настройка готова. Теперь вы можете обращаться к созданной настройке из программы. Предположим, мы хотим, чтобы фон формы имел цвет, определенный в настройке под именем **BackgroundColor**. Тогда нужно в обработчике события формы `Load` записать код, приведенный в листинге 9.26.

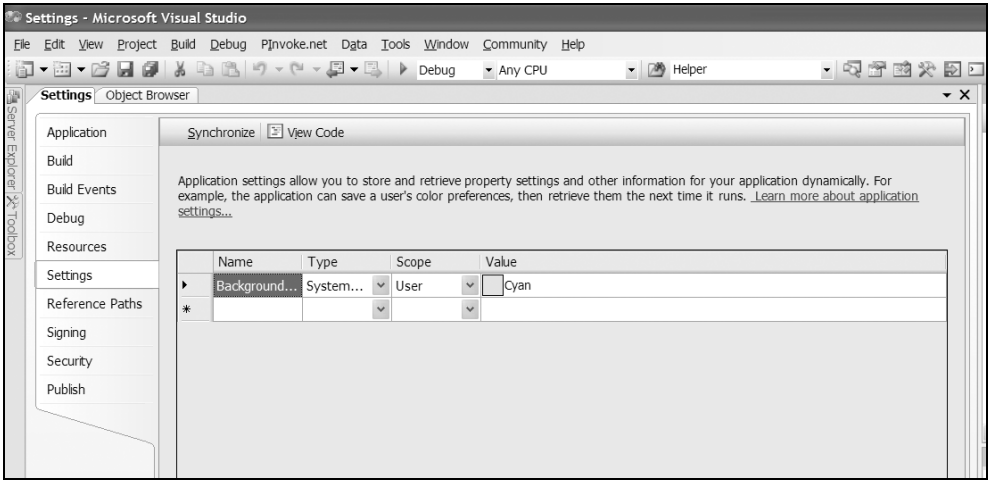



**Рис. 9.3.** Окно дизайнера настроек

**Листинг 9.26. Использование настроек**

```
private void Form1_Load(object sender, EventArgs e)
{
    this.BackColor = Properties.Settings.Default.BackgroundColor;
}
```

Как видите, для доступа к нужной настройке мы обращаемся к пространству имен `Properties`, которое в объекте `Properties.Settings.Default` содержит все настройки, определенные нами в дизайнере настроек. Наверное, вы обратили внимание, что при вводе кода технология IntelliSence услужливо предлагала список возможных значений. Так как мы определили настройку как **User**, то можем не только считывать значение настройки, но и записать новое

значение при необходимости. Добавим на форму кнопку, которая будет менять цвет фона формы на красный (листинг 9.27).

**Листинг 9.27. Установка новых значений**

```
private void button1_Click(object sender, EventArgs e)
{
    Properties.Settings.Default.BackgroundColor = Color.Red;
    Properties.Settings.Default.Save();
}
```

Обратите внимание на вторую строчку. Метод `Save` сохраняет настройку в файле настроек или при последующем запуске программы, наша форма будет окрашена уже в цвет `Red` вместо цвета `Cyan`. Без использования этого метода настройка будет работать только в текущей сессии. Проверьте это самостоятельно. Сами настройки хранятся в файле app.config, который вы можете просмотреть из Visual Studio. Вы увидите, что это обычный XML-файл с записанными нами настройками. Более подробно о настройках можно почитать в документации.

***Примечание***

Пример находится в папке Settings на прилагаемом диске.

# Работа с реестром

Windows хранит море информации о системе, настройках программ, пользователях и множество других данных в специальном хранилище — реестре. Существует возможность улучшить работу программ путем изменения параметров, содержащихся в реестре. В Интернете можно найти целый класс программ, называемых твикерами, в которых собраны различные полезные и интересные настройки, которые можно поменять в реестре, чтобы изменить функциональность программ. Не представляет труда самому написать такой твикер при помощи управляемого кода. В состав .NET Framework входит класс `Microsoft.Win32.Registry`, специально предназначенный для работы с реестром. Рассмотрим один пример, который позволяет изменить заголовок браузера Internet Explorer. По умолчанию при запуске браузера мы видим в заголовке окна название сайта плюс строчку Windows Internet Explorer. Изменим ее, записав новое значение в реестре (листинг 9.28).

**Листинг 9.28. Работа с реестром**

```
using Microsoft.Win32;

private void button1_Click(object sender, EventArgs e)
{
    RegistryKey newIETitle = Registry.CurrentUser.OpenSubKey(
        @"SOFTWARE\Microsoft\Internet Explorer\Main", true);
    newIETitle.SetValue("Window Title","C#. Народные советы");
    newIETitle.Close();
    MessageBox.Show("Закройте IE и запустите его снова");
}
```

Запустив снова браузер Internet Explorer, вы увидите, что в заголовке окна теперь отображается установленная нами строка (рис. 9.4). На сайте **http://winchanger.whatis.ru/** вы можете найти Справочник по реестру, в котором собрано несколько сотен описаний настроек реестра. Вооружившись этим справочником, вы можете продолжить опыты с реестром.

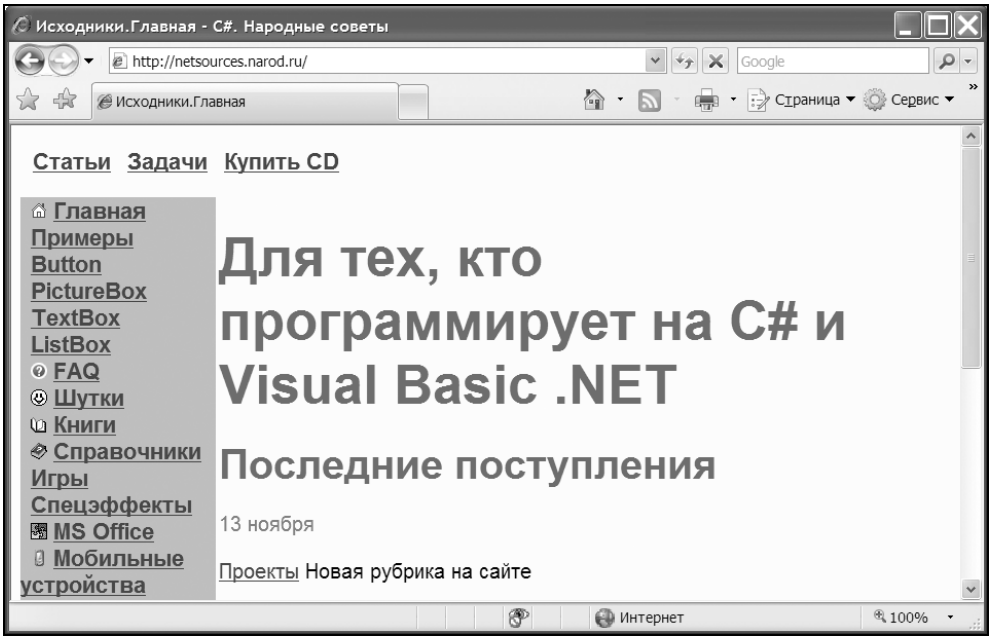



**Рис. 9.4.** Internet Explorer с измененной строкой заголовка

## Определение архитектуры операционной системы

С помощью реестра, например, можно определить, является ли операционная система 32-битной или 64-битной. В будущем Microsoft обещает, что все операционные системы будут только 64-битными. А пока мы находимся в переходном периоде, когда существуют разные архитектуры операционных систем — стандартная 32-битная и более прогрессивная 64-битная, имеющая некоторые дополнительные возможности. Если программист собирается использовать дополнительные возможности 64-битной архитектуры, то он сначала должен определить версию используемой системы. Это можно сделать с помощью реестра, прочитав нужное значение параметра, как показано в листинге 9.29. Впервые я натолкнулся на этот способ на блоге dirtyDogStink (**http://dirtydogstink.com/blog/default.aspx**).

**Листинг 9.29. Определение архитектуры операционной системы**

```
private string GetProcessorArchitecture()
{
    RegistryKey environmentKey = Registry.LocalMachine;  // раздел HKLM
    environmentKey = environmentKey.OpenSubKey(
        @"System\CurrentControlSet\Control\Session Manager\Environment",
        false);
    string strEnvironment =
        environmentKey.GetValue("PROCESSOR_ARCHITECTURE").ToString();
    return strEnvironment;
}

private void butDetectBitVersion_Click(object sender, EventArgs e)
{
    MessageBox.Show(GetProcessorArchitecture());
}
```

## Добавление программы в автозагрузку

Напишем программу, которая будет добавлять себя в автозапуск Windows с помощью реестра. Чтобы программа запускалась при загрузке Windows, необходимо создать ключ реестра в ветке `HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows/CurrentVersion/Run`. Названием ключа будет имя нашей программы, а значением — путь к исполняемому файлу программы.

Как уже писалось выше, в C# за работу с реестром отвечает класс `Microsoft.Win32.Registry`, а за работу с ключами — класс `Microsoft.Win32.RegistryKey`. Расположим на форме две кнопки: **Добавить** (будет создавать ключ в реестре) и **Удалить** (удаляет ключ). В обработчике для кнопки **Добавить** напишем код, приведенный в листинге 9.30.

**Листинг 9.30. Добавление программы в автозагрузку**

```
private void button1_Click(object sender, EventArgs e)
{
    RegistryKey myKey =         Registry.LocalMachine.OpenSubKey(
        @"SOFTWARE\Microsoft\Windows\CurrentVersion\Run", true);
    myKey.SetValue("MyProgram", Application.ExecutablePath);
}
```

Проверим работу кнопки. Нажмем на кнопку **Добавить**, после чего запустим редактор реестра regedit.exe, доберемся до ветки `Run` и проверим, создан ли ключ. Если он создан, то мы можем перезагружать компьютер, и наша программа запустится при загрузке Windows. Следует отметить, что первый параметр функции `SetValue` — это название ключа, которое в данном случае не играет большой роли. А вот второй параметр — значение, которое должно быть установлено равным пути к исполняемому файлу нашей программы.

Для удаления ключа в обработчике нажатия кнопки напишем несложный код, приведенный в листинге 9.31.

**Листинг 9.31. Удаление программы из автозагрузки**

```
private void button2_Click(object sender, EventArgs e)
{
    RegistryKey myKey = Registry.LocalMachine.OpenSubKey(
        @"SOFTWARE\Microsoft\Windows\CurrentVersion\Run", true);
    myKey.DeleteValue("MyProgram");
}
```

Стоит отметить, что в качестве параметра метода `DeleteValue` мы указываем наименование ключа, который нужно удалить из текущей ветки реестра, то есть то наименование, которое мы записали в реестр. Снова запускаем regedit.exe и проверяем, что ключ удалился. Вот такие несложные строки кода позволяют программистам добавлять свои программы в автозапуск.

***Примечание***

Пример работы с реестром находится в папке RegistryDemo на прилагаемом диске.

# Получение информации об изменениях в системе

Класс `SystemEvents` в пространстве имен `Microsoft.Win32` позволяет установить обработчики 14 событий, сообщающих вам об изменениях на уровне системы.

## Как узнать, что пользователь изменил разрешение экрана?

Класс `SystemEvents` из пространства имен `Win32` имеет в своем составе событие `DisplaySettingsChanged`, с помощью которого можно узнать, что пользователь изменил разрешение экрана (листинг 9.32).

**Листинг 9.32. Определение смены разрешения экрана**

```
using Microsoft.Win32;

public Form1()
{
    InitializeComponent();
    SystemEvents.DisplaySettingsChanged +=
        new EventHandler(DisplaySettingsChanged);
}

void DisplaySettingsChanged(object obj, EventArgs ea)
{
    MessageBox.Show("Вы изменили разрешение экрана");
}
```

В составе .NET Framework 2.0 появилось новое дополнительное событие `DisplaySettingsChanging`, которое возникает перед изменением разрешения экрана. Его использование показано в листинге 9.33.

**Листинг 9.33. Событие `DisplaySettingsChanging`**

```
SystemEvents.DisplaySettingsChanging +=
    new EventHandler(DisplaySettingsChanging);


void DisplaySettingsChanging(object obj, EventArgs ea)
{
    MessageBox.Show("Разрешение экрана изменилось");
}
```

## Изменение времени

Рассмотрим еще одно событие `TimeChanged` (листинг 9.34), которое возникает при смене системного времени через приложение Панели управления **Дата и Время**.

**Листинг 9.34. Отслеживание изменения времени**

```
SystemEvents.TimeChanged += new EventHandler(SystemEvents_TimeChanged);

void SystemEvents_TimeChanged(object sender, EventArgs e)
{
    this.Text = "Вы зачем поменяли системное время?";
}
```

Когда вы поменяете время в Панели управления и нажмете на кнопку **Применить**, то программа получит сообщение через событие `TimeChanged`.

Остальные системные события изучите самостоятельно.

Также обратите внимание, что к событиям `SessionEnded` и `SessionEnding` в .NET Framework 2.0 добавилось новое событие `SessionSwitch`, которое происходит при переключении на другую учетную запись системы.

> ***Примечание***
>
> Примеры работы с классом `SystemEvents` находятся в папке SystemEventsDemo на прилагаемом диске.

# Консольные приложения

В .NET Framework 2.0 появились новые методы и свойства, позволяющие работать с консолью. Теперь нет нужды использовать такие вызовы функций Windows API, как `SetConsoleTitle`, `SetConsoleTextAttribute`, `GetConsoleScreenBufferInfo` и др. Все стало гораздо проще: появился специальный класс `Console`. С помощью его свойств `BackgroundColor` и `ForegroundColor` можно поменять цвет символов и фона, а с помощью свойства `Title` легко меняется заголовок консольного окна. Кроме того, появились такие интересные свойства, как `CapsLock` и `NumberLock`, которые позволяют определить состояние соответствующих индикаторов на клавиатуре. В листинге 9.35 приведен простой пример создания цветного консольного приложения.

**Листинг 9.35. Консольное приложение**

```
using System;

class AdvancedConsoleDemo
{
    static void Main()
    {
        Console.Title = "C#.НародныеСоветы.Консольная программа";
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine("Цветной текст");

        if (Console.CapsLock)
          Console.WriteLine("Caps Lock включен");
        else
          Console.WriteLine("Caps Lock выключен");

        Console.WindowHeight = 20;
        Console.WindowWidth = 40;
        Console.ResetColor();
        Console.ReadLine();
    }
}
```

***Примечание***

Пример консольного приложения находится в папке ColorConsole на прилагаемом диске.

# Журналы событий

Журналы событий являются удобными средствами для записи информации о ходе работы программы или сервиса.

С помощью класса `EventLog` можно легко получить доступ к журналу событий Windows или настроить его под свои нужды. Вы можете считать данные из журнала событий, сделать там запись, создать или удалить источники событий и многое другое. Особый интерес для разработчика данный класс представляет по той причине, что позволяет работать как с локальными, так и с удаленными компьютерами.

## Как найти доступные журналы событий на компьютере?

Начнем с простого. Для перечисления всех доступных журналов событий нужно вызвать метод `GetEventLogs` класса `EventLog`. Метод `GetEventLogs` выполняет поиск всех журналов событий на заданном компьютере и создает массив объектов `EventLog`, содержащий список журналов. Его применение проиллюстрировано в листинге 9.36.

**Листинг 9.36. Получение списка журналов событий на локальном компьютере**

```
EventLog[] eventLogs;
// Получим список журналов событий для локального компьютера
eventLogs = EventLog.GetEventLogs(System.Environment.MachineName);
this.Text = "Число журналов событий: " + eventLogs.Length;

for (int i = 0; i < eventLogs.Length; i++)
    listBox1.Items.Add(eventLogs[i].Log);
```

Для получения списка журналов событий удаленного компьютера MyServer используйте строку

```
eventLogs = EventLog.GetEventLogs("MyServer");
```

## Чтение и запись логов в журнал событий

Для чтения записей журнала событий используется свойство `Entries` класса `EventLog`. Свойство `Entries` является коллекцией всех существующих запи-

сей в журнале. Вам нужно пройтись по всей коллекции и извлечь необходимые данные, как показано в листинге 9.37.

**Листинг 9.37. Чтение записей из журнала событий**

```
 //logType может быть Application, Security, System или
// собственный журнал событий
string logType = "System";

EventLog ev = new EventLog(logType, System.Environment.MachineName);
int LastLogToShow = ev.Entries.Count;
if (LastLogToShow <= 0)
    MessageBox.Show("Нет записей в журнале событий: " + logType);
else
{
    // Прочитать 2 последние записи в указанном журнале.
    int i;
    for (i = ev.Entries.Count - 1; i >= LastLogToShow - 2; i--)
    {
        EventLogEntry CurrentEntry = ev.Entries[i];
        listBox1.Items.Add("Event ID : " + CurrentEntry.InstanceId);
        listBox1.Items.Add("Entry Type : " +
            CurrentEntry.EntryType.ToString());
        listBox1.Items.Add("Message :  " + CurrentEntry.Message);
    }
    ev.Close();
}
```

***ПРИМЕЧАНИЕ***

`InstanceId` является новым свойством, которое появилось в .NET Framework 2.0 и заменило собой устаревшее свойство `EventID`.

## Запись в журнал

Для записи в журнал событий используется метод `WriteEntry` класса `EventLog` (листинг 9.38). Необходимо помнить, что у вас должны быть права для записи.

**Листинг 9.38. Запись в журнал событий**

```
// Создадим источник, если он не существует
if (!EventLog.SourceExists("MySource"))
{
    EventLog.CreateEventSource("MySource", "MyNewLog");
}

EventLog myLog = new EventLog();
myLog.Source = "MySource";

// Делаем запись в журнал событий
myLog.WriteEntry("Сделана запись.");
```

## Очистка записей в журнале событий

Когда журнал событий заполнен записями, то запись новых событий прекращается или происходит замещение ранних записей. Вы можете очистить журнал событий, чтобы начать запись информации заново. Для этих целей существует метод `Clear`. Для выполнения операции, приведенной в листинге 9.39, вы должны иметь права администратора.

**Листинг 9.39. Очистка записей в журнале событий**

```
// Создаем экземпляр EventLog и передаем имя журнала событий и имя машины
EventLog ev = new EventLog("Security", System.Environment.MachineName);
ev.Clear();
ev.Close();
```

## Создание собственного журнала событий

Как вы могли заметить из листинга 9.38, метод `CreateEventSource` можно использовать для создания собственного журнала событий. Перед созданием нужно проверить существование журнала событий и только затем вызвать метод `CreateEventSource`. В противном случае вы получите ошибку `System.ArgumentException`. В листинге 9.40 показано альтернативное использование метода `CreateEventSource` с классом `EvenSourceCreationData`.

**Листинг 9.40. Создание собственной записи в журнале событий**

```
// Создаем источник, если он еще не существует
if (!(EventLog.SourceExists("MySour", System.Environment.MachineName)))
{
    EventSourceCreationData mySourceData =
        new EventSourceCreationData("MySour", "myLog");
    EventLog.CreateEventSource(mySourceData);
    MessageBox.Show("CreatingEventSource");
}
```

## Удаление собственного журнала событий

Для удаления журнала событий используется метод `Delete` класса `EventLog` (листинг 9.41). При удалении собственного журнала событий удостоверьтесь, что другие источники не делают записей в этом журнале.

**Листинг 9.41. Удаление собственной записи из журнала событий**

```
string logName = "MyLog";
if (EventLog.SourceExists("MyOldSource", System.Environment.MachineName))
{
    logName = EventLog.LogNameFromSourceName("MySour",
                  System.Environment.MachineName);
    EventLog.DeleteEventSource("MySour", System.Environment.MachineName);
    EventLog.Delete(logName, System.Environment.MachineName);
    MessageBox.Show(logName + " удален.");
}
```

***ПРИМЕЧАНИЕ***

Примеры работы с журналами событий находятся в папке EventLogDemo на прилагаемом диске.

# Измерение времени выполнения кода в приложении

Чтобы оценить скорость работы какого-нибудь кода в приложении, нужно иметь определенную методику. Самый простой способ — это использовать

методы `TickCount` из класса `Environment` или `Ticks` из класса `DateTime`. Но эти способы не блещут точностью. Рассмотрим все варианты.

## Измерение с помощью функций Windows API

На сайте MSDN имеется статья, в которой говорится, как измерять выполнение кода с очень высокой точностью (адрес статьи **http://support.microsoft.com/kb/306979**). В листинге 9.42 приведен код из этой статьи, показывающий, как это делать.

**Листинг 9.42. Измерение с помощью функций Windows API**

```
[DllImport("kernel32.dll")]
extern static short QueryPerformanceCounter(ref long lpPerformanceCount);
[DllImport("kernel32.dll")]
extern static short QueryPerformanceFrequency(ref long lpFrequency);

private void Form1_Load(object sender, EventArgs e)
{
    long ctr1 = 0, ctr2 = 0, freq = 0;
    int acc = 0, i = 0;

    if (QueryPerformanceCounter(ref ctr1) != 0) // Начинаем отчет времени
    {
        for (i = 0; i < 1000; i++) acc++;   // выполняем действия
        QueryPerformanceCounter(ref ctr2);  // заканчиваем отчет времени.
        lblInfo.Text = "Начало: " + ctr1 + Environment.NewLine;
        lblInfo.Text = lblInfo.Text + "Конец: " +
            ctr2 + Environment.NewLine;
        QueryPerformanceFrequency(ref freq);
        lblInfo.Text = lblInfo.Text +
            "Минимальное разрешение QueryPerformanceCounter: 1/" +
            freq + " сек." + Environment.NewLine;
        lblInfo.Text = lblInfo.Text +
            "Время инкрементирования до 1000: " +
            (ctr2 - ctr1) * 1.0 / freq + " сек.";
    }
    else
        lblInfo.Text = "Счетчик высокой точности не поддерживается.";
}
```

Данный код обеспечивает точность до 1/3579545 секунды.

***ПРИМЕЧАНИЕ***

Пример находится в папке QueryPerformanceDemo на прилагаемом диске.

## Измерение с помощью метода *Ticks*

Другой способ (листинг 9.43), который обеспечивает среднюю точность, предполагает использование свойства `Ticks` класса `DateTime`.

**Листинг 9.43. Измерение с помощью метода `Ticks`**

```
// Начало измерения
long startTicks = DateTime.Now.Ticks;
// Операция, которую нужно замерить
… ваш код
long endTicks = DateTime.Now.Ticks; // завершение измерения
long delta = endTicks – startTicks;

MessageBox.Show("Ticks = " + delta.ToString());
// Конвертируем в секунды
delta = delta/(long)10000000;     // один такт равен 100 наносекундам
MessageBox.Show("Операция длилась: " + delta.ToString());
```

Данный способ обеспечивает точность до 100 наносекунд.

## Измерение с помощью *TickCount*

Наименьшую точность обеспечивает пример с использованием свойства `TickCount`, который приведен в листинге 9.44.

**Листинг 9.44. Измерение с помощью `TickCount`**

```
// Начало измерения
int startOp = Environment.TickCount;
// Операция, которую нужно замерить
… ваш код
```

```
int endOp = Environment.TickCount; //завершение измерения
// Выводим результат
MessageBox.Show("TickCount = " + (endOp - startOp));
```

Итак, вы рассмотрели все три способа и выбрали самый очевидный вариант с применением функции Windows API? Не торопитесь. Вот что думает по этому поводу в своем блоге **http://blogs.gotdotnet.ru/personal/mihailik** Олег Михайлик. Приведу часть его аргументации.

Метод с использованием функций Windows измеряет время очень точно. Но зачем вообще может понадобиться такая точность? Допустим, мы хотим измерить скорость очень быстрой операции. Как бы вы поступили в этом случае? Я думаю, для грамотного инженера ответ очевиден: запустить сто, тысячу, миллион операций — и измерить общее время выполнения.

Очевидно, что метод "цикла" значительно надежнее и точнее самого лучшего в мире измерения, но единичной операции. Это очевидные статистические законы.

Если мы применим метод с использованием `QueryPerformance` для измерения продолжительности того же большого цикла, то он сразу потеряет все свои преимущества. Если операция длится секунды, то погрешность в третьем знаке не играет значения. На фоне длительных операций использование `DateTime.Now` практически ничем не хуже системных счетчиков производительности. Также нельзя обойти стороной проблему архитектуры процессора. Многие современные и все будущие процессоры — двуядерные. Следовательно, измерять такты процессора не так уж и правомочно. Измеряемая операция попросту может выполняться на нескольких процессорах последовательно, "перепрыгивая" с одного на другой. Кроме того, многие современные процессоры могут на лету изменять частоту. Если за время измерения этот показатель изменится, точность сразу пострадает. Отсюда вывод — для измерений ЛУЧШЕ использовать более простой и прозрачный метод `DateTime.Now`.

## Класс *StopWatch*

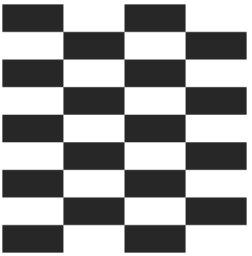
Казалось бы, в дискуссии по измерению длительности операции поставлена точка. Но существует еще и четвертый способ! В .NET Framework 2.0 появился новый класс `StopWatch`, который, как сказано в документации MSDN, и

призван заменить неуправляемый код с использованием функций `QueryPerformanceCounter` и `QueryPerformanceFrequency`. Изучением данного класса займитесь самостоятельно.

## Заключение

Итак, мы рассмотрели типичные примеры работы с приложениями. Теперь вы можете приступать к созданию сложных программ, которых так ждут ваши пользователи. В этой главе я обошел вниманием такие важные вещи, как отладка приложений, поиск багов, тестирование. Но здесь простыми трюками не обойтись. Необходимо почитать серьезную литературу, чтобы стать хорошим специалистом по этим вопросам.

## Глава 10



# Диски, папки и файлы

В этой главе мы поговорим о файловой системе. Начнем с дисков, потом рассмотрим приемы работы с папками и закончим главу советами по работе с файлами.

## Диски

Диски являются вершиной в иерархии файловой системы, и именно в них содержатся файлы и папки. Рассмотрим наиболее востребованные программистами приемы извлечения нужной информации.

### Как получить список логических дисков?

В .NET Framework имеются два метода `GetLogicalDrives`, которые возвращают массив имен логических дисков. Один метод относится к классу `Directory`, а второй — к классу `Environment`. В листинге 10.1 рассматриваются оба варианта.

**Листинг 10.1. Получение списка логических дисков. Два способа**

```
private void button1_Click(object sender, EventArgs e)
{
    string[] astrLogicalDrives = System.IO.Directory.GetLogicalDrives();
    foreach (string disk in astrLogicalDrives)
        listBox1.Items.Add(disk);
}

private void button2_Click(object sender, EventArgs e)
{
```

```
    string[] astrLogicalDrives = System.Environment.GetLogicalDrives();
    foreach (string disk in astrLogicalDrives)
        listBox1.Items.Add(disk);
}
```

Разница между этими одноименными методами в том, что они вызывают разные типы исключений, связанные с правами доступа и разрешениями.

## Как узнать тип диска и его свойства?

Несмотря на растущие объемы жестких дисков, наступает момент, когда пользователь начинает поглядывать на размер свободного места на своих дисках. Фотографии, фильмы, музыка, "навороченные" программы быстро заполняют диски. Чтобы узнать количество свободного места на диске, можно воспользоваться свойством `DriveInfo.AvailableFreeSpace`. Обратите внимание, что класс `DriveInfo` является новым классом в .NET Framework 2.0. Поэтому вам, возможно, придется пересмотреть старые проекты, в которых вы получали информацию о дисках при помощи неуправляемого кода. Для создания нового объекта `DriveInfo` нужно указать букву диска или более привычную запись с двоеточием и обратной косой чертой, например, `"C"` или `"C:\"` для диска C:. В листинге 10.2 приведен пример, показывающий, как извлечь информацию о нужном диске.

**Листинг 10.2. Получение типа диска и его свойств**

```
private void button1_Click(object sender, EventArgs e)
{
    // Выводим информацию о диске
    System.IO.DriveInfo drv = new System.IO.DriveInfo(@"d:\");
    listBox1.Items.Clear();
    listBox1.Items.Add("Диск: " + drv.Name);

    if (drv.IsReady)
    {
        listBox1.Items.Add("Тип диска: " + drv.DriveType.ToString());
        listBox1.Items.Add("Файловая система: " +
                           drv.DriveFormat.ToString());
        listBox1.Items.Add("Свободное место на диске: " +
                           drv.AvailableFreeSpace.ToString());
    }
}
```

Надо отметить, что у класса `DriveInfo` есть еще свойство `TotalFreeSpace`, изучить которое я предлагаю самостоятельно. Обратите внимание, что свойство `AvailableFreeSpace` заменило функцию Windows API `GetFreeDiskSpaceEx`. Хороший пример использования этой функции Windows API можно найти на сайте CodeProject (**http://www.codeproject.com/useritems/tips/Freespace_src.zip**). Но в новых проектах, построенных на платформе .NET Framework 2.0, использовать этот пример нет никакой необходимости.

> ***Примечание***
>
> Примеры, иллюстрирующие работу с дисками, находятся в папке LogicalDisk на прилагаемом компакте.

# Папки

Основные операции, проводимые с папками, — это создание, переименование, перемещение и удаление. У класса `System.IO.Directory` имеются соответствующие методы `CreateDirectory`, `Move` и `Delete`, которые позволяют легко выполнить эти задачи. Рассмотрим другие методы для работы с папками.

## Как получить список папок?

Получить список папок можно также несколькими способами. Например, можно воспользоваться методом `GetDirectories` класса `System.IO.Directory`. Как это сделать, показано в листинге 10.3.

**Листинг 10.3. Список папок**

```
private void button1_Click(object sender, EventArgs e)
{
    // Получим список папок на диске D:
    listBox1.Item.Clear();
    string[] astrFolders = System.IO.Directory.GetDirectories(@"d:\");
    foreach(string folder in astrFolders)
        listBox1.Items.Add(folder);
}
```

Также метод `GetDirectories` имеется в классе `DirectoryInfo`. Рассмотрим и этот вариант, и при этом выведем список папок по маске (листинг 10.4). Например, мы хотим получить список только тех папок, в именах которых встречается буквосочетание "pro".

**Листинг 10.4. Список папок по маске**

```
private void button2_Click(object sender, EventArgs e)
{
    // Получим список папок, где встречается буквосочетание pro
    listBox1.Items.Clear();
    System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(@"d:\");
    System.IO.DirectoryInfo[] folders = di.GetDirectories("*pro*");
    foreach (System.IO.DirectoryInfo maskdirs in folders)
        listBox1.Items.Add(maskdirs);
}
```

## Как проверить существование папки?

Для этого существует метод `System.IO.Directory.Exists`, который возвращает значение `true`, если папка существует (листинг 10.5).

**Листинг 10.5. Проверка существования папки**

```
if (System.IO.Directory.Exists(@"C:\windows"))
      label1.Text = "Папка " + @"C:\Windows" + " существует";
else
      label1.Text = "Папка не существует";
```

## Как переименовать папку?

Если вам нужно переименовать папку, то воспользуйтесь методом `Move` класса `Directory` (листинг 10.6). С точки зрения системы, переименование папки является перемещением папки в ее же родительскую папку, но под другим именем.

**Листинг 10.6. Переименование папки**

```
using System.IO;

// Переименовываем папку MyFolder в папку NewFolder
string oldPathString = @"C:\MyFolder";
string newPathString = @"C:\NewFolder";
Directory.Move(oldPathString, newPathString);
```

## Как удалять папки?

Хотелось бы обратить ваше внимание на случай с удалениями папок. Метод `Directory.Delete` удаляет только пустую папку. Если в папке содержатся подпапки или файлы, то вы получите ошибку. В листинге 10.7 приведен код для проверки, что поведение именно таково.

**Листинг 10.7. Удаление пустой папки**

```
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        // Попытаемся удалить папку C:\WUTEMP
        System.IO.Directory.Delete(@"c:\wutemp");
        MessageBox.Show("Папка удалена.");
    }
    catch (Exception)
    {
       MessageBox.Show("Нельзя удалить непустую папку.");
    }
    finally {}
}
```

Поэтому для удаления папок с вложенными папками и файлами используйте перегруженную версию `Directory.Delete(path, recursive)`, которая позволит удалить папку со всеми вложенными файлами и подпапками.

## Как выбрать папку?

Начиная с версии 1.1 (Visual Studio .NET 2003) .NET Framework содержит класс `FolderBrowserDialog` из пространства имен `System.Windows.Forms`, который служит для работы с диалоговым окном выбора папки (аналог функции Windows API `SHBrowserFolder`). Данный класс представляет удобный диалог для выбора нужной папки, а также, в случае необходимости, может предоставить пользователю возможность создания новой папки. Его использование показано в листинге 10.8.

**Листинг 10.8. Выбор папки**

```
private void butShowFolderDialog_Click(object sender, EventArgs e)
{
    FolderBrowserDialog fbd = new FolderBrowserDialog();
    // задаем папку верхнего уровня
    fbd.RootFolder = Environment.SpecialFolder.MyComputer;
    // Заголовок в диалоговом окне
    fbd.Description = "Выберите папку";
    // Не выводим кнопку Новая папка
    fbd.ShowNewFolderButton = false;
    // Получаем папку, выбранную пользователем
    if (fbd.ShowDialog() == DialogResult.OK)
        this.Text = fbd.SelectedPath;
}
```

## Как получить путь для папки Мои документы и других специальных папок Windows?

В листинге 10.9 показано, как использовать метод `GetFolderPath` класса `System.Environment` для получения этой информации.

**Листинг 10.9. Получение путей к специальным папкам**

```
// Получим путь к папке Мои документы
MessageBox.Show( Environment.GetFolderPath(
                                Environment.SpecialFolder.Personal ) );
```

В число специальных папок входят следующие папки:

- `ApplicationData` — каталог, в котором резервируются данные приложения текущего подвижного пользователя (то есть такого пользователя, который может работать на нескольких компьютерах сети, и чей профиль хранится на сервере сети и загружается в систему при подключении пользователя);
- `CommonApplicationData` — каталог, выполняющий функции общего хранилища данных приложения, используемого всеми пользователями;
- `CommonProgramFiles` — каталог для компонентов, общих для приложений;
- `Cookies` — каталог, в котором хранятся файлы cookies;
- `Desktop` — логический рабочий стол;
- `DesktopDirectory` — каталог, используемый для физического хранения объектов файла рабочего стола (следует различать этот каталог и папку рабочего стола, которая является виртуальной папкой);
- `Favorites` — каталог, в котором хранятся ссылки на Избранное;
- `History` — каталог, в котором хранятся элементы журнала Интернета;
- `InternetCache` — каталог, содержащий временные файлы Интернета;
- `LocalApplicationData` — каталог, выполняющий функции общего хранилища данных приложения, используемых текущим неподвижным пользователем;
- `MyComputer` — папка Мой компьютер;
- `MyMusic` — папка Моя музыка;
- `MyPictures` — папка Мои фотографии;
- `Personal` — каталог, служащий в качестве общего хранилища для документов;
- `ProgramFiles` — каталог файлов программ;
- `Programs` — каталог, содержащий группы программ пользователя;
- `Recent` — каталог, содержащий последние используемые документы;
- `SendTo` — каталог, содержащий ярлыки, отображаемые в пункте меню **Отправить**;
- `StartMenu` — каталог, содержащий пункты меню **Пуск**;
- `Startup` — каталог, соответствующий группе программ автозагрузки пользователя;

- `System` — системный каталог (обычно C:\Windows);
- `Templates` — каталог, в котором хранятся шаблоны документов.

Обратите внимание, что в новой операционной системе Windows Vista изменились некоторые названия папок. Например, вместо названия *Мой компьютер* используется просто *Компьютер*. Вот еще небольшой список изменившихся названий. Сначала приводится старое стандартное название специальной папки, а затем следует новое расположение папки в Windows Vista:

- Application Data — AppData\Roaming;
- Cookies — \AppData\Roaming\Microsoft\Windows\Cookies;
- Local Settings — \AppData\Local;
- My Documents — \Documents;
- NetHood — \AppData\Roaming\Microsoft\Windows\Network Shortcuts;
- PrintHood — \AppData\Roaming\Microsoft\Windows\Printer Shortcuts;
- Recent — \AppData\Roaming\Microsoft\Windows\Recent;
- SendTo — \AppData\Roaming\Microsoft\Windows\SendTo;
- Start Menu — \AppData\Roaming\Microsoft\Windows\Start Menu;
- Templates — \AppData\Roaming\Microsoft\Windows\Templates.

Хочу обратить особое внимание, что вместо папки Documents and Settings, в которой хранились данные пользователей системы, появилась новая папка Users. Поэтому, если в ваших проектах использовались жестко прошитые названия этой папки, то переработайте свой код. Кстати, я нашел неплохую статью на эту тему "Finding the User Settings in Vista" по адресу **http://www.devsource.com/c/a/Using-VS/Finding-the-User-Settings-in-Vista/**.

## Свойства папки

В листинге 10.10 приведен пример того, как можно узнать подробную информацию о свойствах папки.

**Листинг 10.10. Получение свойств папки**

```
private void button4_Click(object sender, EventArgs e)
{
    // Получим информацию о свойствах папки
```

```
    System.IO.DirectoryInfo dir = new
                        System.IO.DirectoryInfo(@"c:\wutemp");
    listBox1.Items.Clear();
    listBox1.Items.Add("Проверка папки: " + dir.Name);
    listBox1.Items.Add("Родительская папка: " + dir.Parent.Name);
    listBox1.Items.Add("Папка существует: ");
    listBox1.Items.Add(dir.Exists.ToString());
    if (dir.Exists)
    {
        listBox1.Items.Add("Папка создана: ");
        listBox1.Items.Add(dir.CreationTime.ToString());
        listBox1.Items.Add("Папка изменена: ");
        listBox1.Items.Add(dir.LastWriteTime.ToString());
        listBox1.Items.Add("Время последнего доступа: ");
        listBox1.Items.Add(dir.LastAccessTime.ToString());
        listBox1.Items.Add("Атрибуты папки: ");
        listBox1.Items.Add(dir.Attributes.ToString());
        listBox1.Items.Add("Папка содержит: " +
        dir.GetFiles().Length.ToString() + " файла");
    }
}
```

## Размеры папки

Если вам необходимо вычислить размер папки, включая подпапки, то придется написать небольшую процедуру (листинг 10.11), которая займется вычислениями размеров всех файлов в папке. Для удобства сделаем процедуру универсальной, чтобы она могла вычислять размеры как самой папки без учета файлов в подпапках, так и с учетом всех подпапок.

**Листинг 10.11. Получение размеров папки**

```
static long GetDirectorySize(System.IO.DirectoryInfo directory,
                             bool includeSubdir)
{
      long totalSize = 0;
      // Считаем все файлы
      System.IO.FileInfo[] files = directory.GetFiles();
      foreach (System.IO.FileInfo file in files)
      {
          totalSize += file.Length;
      }
```

```
        // Считаем все подпапки
        if (includeSubdir)
        {
            System.IO.DirectoryInfo[] dirs = directory.GetDirectories();
            foreach (System.IO.DirectoryInfo dir in dirs)
            {
                totalSize += GetDirectorySize(dir, true);
            }
        }
        return totalSize;
    }

private void butFolderSize_Click(object sender, EventArgs e)
{
    this.Cursor = Cursors.WaitCursor;
    System.IO.DirectoryInfo dir = new
                        System.IO.DirectoryInfo(@"d:\help");
    textBox1.Text = "Общий размер: " +
    GetDirectorySize(dir, true).ToString() + " байт.";
    this.Cursor = Cursors.Default;
}
```

Во время длительной операции обязательно переключайте указатель мыши, чтобы он имел вид так называемых песочных часов (`WaitCursor`). В данном случае вычисление размера папки может занять значительное время и у пользователя возникнет ощущение, что программа просто зависла.

> ***Примечание***
>
> Примеры работы с папками находятся в папке DirectoryDemo на прилагаемом диске.

## Как написать свой Проводник?

В принципе, на основе предыдущих советов вам под силу написать свой собственный Проводник, точнее его левую часть, в которой отображается древовидная структура папок (рис. 10.1). Для этого расположите на форме элемент управления `TreeView` и установите его свойство `Dock` в значение `Fill`. Далее добавьте код, приведенный в листинге 10.12.

**Листинг 10.12. Создание древовидной структуры папок**

```
using System.IO;

private void Form1_Load(object sender, EventArgs e)
{
    // Заполняем первый узел. Используем диск C:
    TreeNode rootNode = new TreeNode(@"C:\");
    treeDirectory.Nodes.Add(rootNode);
    // Заполнеяем первый уровень и раскрываем его
    FillNodes(rootNode);
    treeDirectory.Nodes[0].Expand();
    // Заполняем второй узел. Используем диск D:
    TreeNode rootNode2 = new TreeNode(@"D:\");
    treeDirectory.Nodes.Add(rootNode2);
    FillNodes(rootNode2);
    // Не будем раскрывать его
    //treeDirectory.Nodes[1].Expand();
}

private void FillNodes(TreeNode dirNode)
{
   DirectoryInfo dir = new DirectoryInfo(dirNode.FullPath);

   foreach (DirectoryInfo dirItem in dir.GetDirectories())
   {
        // Добавляем узел для каждой папки
        TreeNode newNode = new TreeNode(dirItem.Name);
        dirNode.Nodes.Add(newNode);
        newNode.Nodes.Add("*");
    }
}

private void treeDirectory_BeforeExpand(object sender,
                                        TreeViewCancelEventArgs e)
{
    // Если найден узел со звездочкой *, то удаляем его
    // и получаем список подпапок.
    if (e.Node.Nodes[0].Text == "*")
    {
        e.Node.Nodes.Clear();
        FillNodes(e.Node);
    }
}
```

В этом примере мы ограничились двумя дисками, C: и D:, причем не проверяли наличие последнего. В реальных приложениях надо, разумеется, получать список дисков динамически, используя технику, описанную в *разд. "Диски"* этой главы.

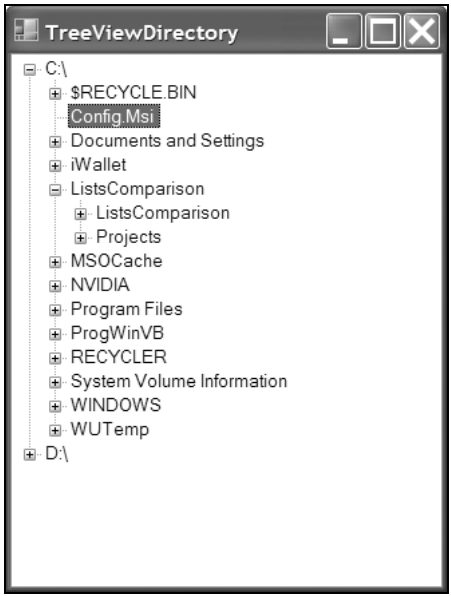



**Рис. 10.1.** Древовидная структура папок

### *Примечание*

Пример создания древовидной структуры папок находится в папке TreeViewDirectory на прилагаемом диске.

# Файлы

Работать с файлами в .NET Framework так же просто, как и с папками. Основные операции осуществляются при помощи методов классов `System.IO.FileInfo` и `System.IO.File` — `CopyTo`, `Create`, `Delete`, `MoveTo` и др. Мы рассмотрим лишь часть приемов, использующихся для работы с файлами.

## Как получить список файлов в папке?

Как и в случае с дисками и папками, существует несколько способов получить список имен файлов в заданной папке. Рассмотрим вариант, в котором основная работа производится методом `GetFiles` класса `Directory` пространства имен `System.IO` (листинг 10.13).

**Листинг 10.13. Получение списка файлов в папке**

```
private void button1_Click(object sender, EventArgs e)
{
    string[] astrFiles = System.IO.Directory.GetFiles(@"c:\");
    listBox1.Items.Add("Всего файлов: " + astrFiles.Length);

    foreach (string file in astrFiles)
        listBox1.Items.Add(file);
}
```

Также список файлов можно получить при помощи метода `GetFiles` класса `DirectoryInfo`.

## Как получить список папок и файлов?

Мы знаем, как получить отдельно список папок и список файлов. Существует еще метод `GetFileSystemEntries` класса `Directory`, который позволяет получить за один вызов список и папок, и файлов. Его использование проиллюстрировано листингом 10.14.

**Листинг 10.14. Получение списка папок и файлов**

```
private void button7_Click(object sender, EventArgs e)
{
    string[] directoryEntries =
            System.IO.Directory.GetFileSystemEntries(@"c:\windows");

    foreach (string str in directoryEntries)
    {
        listBox1.Items.Add(str);
    }
}
```

## Как получить список файлов по маске?

Разберем еще один пример. Иногда требуется получить имена файлов по заданной маске. В маске можно использовать символы "*" и "?". В следующем примере (листинг 10.15) мы ищем файлы с любым именем и расширением из трех букв, которое начинается на "in".

**Листинг 10.15. Получение списка файлов по маске**

```
private void button2_Click(object sender, EventArgs e)
{
    string[] astrFiles = System.IO.Directory.GetFiles(@"c:\","*.in?");
    listBox1.Items.Add("Всего файлов: " + astrFiles.Length);

    foreach (string file in astrFiles)
        listBox1.Items.Add(file);
}
```

## Как узнать, существует ли файл?

Узнать, существует ли нужный файл на диске, можно при помощи метода `System.IO.File.Exists`, который возвращает значение `true`, если файл существует (листинг 10.16).

**Листинг 10.16. Проверка существования файла**

```
if (System.IO.File.Exists(Application.StartupPath + "\\test.txt"))
    label1.Text = "Файл test.txt существует";
else
    label1.Text = "Файл test.txt не существует";
```

## Как получить имя файла из полного пути файла?

Если вам понадобилось получить имя файла из полного пути файла, то используйте методы `System.IO.Path.GetFileNameWithoutExtension` и `System.IO.Path.GetFileName`. В первом случае мы получим имя файла без расширения, а во втором — результат будет включать и имя файла, и расширение (листинг 10.17).

**Листинг 10.17. Получение имени файла из полного пути**

```
private void button3_Click(object sender, EventArgs e)
{
    // Полный путь к файлу
    string fileNamePath = @"c:\windows\system32\notepad.exe";
    // Имя файла с расширением
    listBox1.Items.Add(System.IO.Path.GetFileName(fileNamePath));
    // Имя файла без расширения
    listBox1.Items.Add(
        System.IO.Path.GetFileNameWithoutExtension(fileNamePath));
}
```

## Как получить расширение файла из полного пути?

Если вам захотелось узнать именно расширение файла, воспользуйтесь методом `System.IO.Path.GetExtension` (листинг 10.18).

**Листинг 10.18. Получение расширения файла**

```
private void button4_Click(object sender, EventArgs e)
{
    // Полный путь к файлу
    string fileNamePath = @"c:\windows\system32\notepad.exe";
    // Получим расширение файла
    listBox1.Items.Add(System.IO.Path.GetExtension(fileNamePath));
}
```

## Как создать, удалить, переместить файл?

Простейшие операции с файлами типа создать, удалить, переместить легко осуществляются с помощью методов `Create`, `Delete` и `Move` класса `File`. Также можно воспользоваться схожими методами класса `FileInfo`. В документации имеются хорошие примеры, поэтому нет смысла дублировать эту информацию.

## Как установить атрибуты у файла?

У файлов имеются атрибуты, которые можно прочитать и менять с помощью методов `GetAttributes` и `SetAttributes` класса `System.IO.File`, как показано в листинге 10.19.

**Листинг 10.19. Установка атрибутов файла**

```
private void button8_Click(object sender, EventArgs e)
{
    // путь к тестовому файлу
    string path = @"c:\WUTEMP\test.txt";

    // если файлы имел атрибут Скрытый
    if ((System.IO.File.GetAttributes(path) &
          System.IO.FileAttributes.Hidden)
                    == System.IO.FileAttributes.Hidden)
    {
        // то устанавливаем атрибут Normal
        System.IO.File.SetAttributes(path,
                        System.IO.FileAttributes.Normal);
        MessageBox.Show("Файл больше не является скрытым", path);

    }
    else // если файл не был скрытым
    {
        // то устанавливаем у файла атрибут Скрытый
        System.IO.File.SetAttributes(path,
                          System.IO.File.GetAttributes(path) |
                          System.IO.FileAttributes.Hidden);
        MessageBox.Show("Файл стал скрытым", path);
    }
}
```

## Свойства файла

Подведем некоторые итоги. Мы уже многое можем узнать о файле. В коде, приведенном в листинге 10.20, мы получим некоторые свойства выбранного файла с помощью класса `FileInfo`.

**Листинг 10.20. Получение свойств файла**

```
private void button15_Click(object sender, EventArgs e)
{
    // Выводим информацию о файле.
    System.IO.FileInfo file = new
                System.IO.FileInfo(@"c:\wutemp\text.txt");
    listBox1.Items.Clear();
    listBox1.Items.Add("Свойства для файла: " + file.Name);
    listBox1.Items.Add("Наличие файла: " + file.Exists.ToString());
    if (file.Exists)
    {
        listBox1.Items.Add("Время создания файла: ");
        listBox1.Items.Add(file.CreationTime.ToString());
        listBox1.Items.Add("Последнее изменение файла: ");
        listBox1.Items.Add(file.LastWriteTime.ToString());
        listBox1.Items.Add("Файл был открыт в последний раз: ");
        listBox1.Items.Add(file.LastAccessTime.ToString());
        listBox1.Items.Add("Размер файла (в байтах): ");
        listBox1.Items.Add(file.Length.ToString());
        listBox1.Items.Add("Атрибуты файла: ");
        listBox1.Items.Add(file.Attributes.ToString());
    }
}
```

## Как извлечь информацию о файле?

Некоторые файлы (например, исполняемые файлы EXE или динамические библиотеки DLL) содержат различную информацию. Если выбрать такой файл в Проводнике и через контекстное меню выбрать пункт **Свойства**, то можно увидеть эти свойства. Можно получить эту информацию и программным путем (листинг 10.21). Обратите внимание, что в этом случае используется пространство имен `System.Diagnostics`, а не `System.IO`, как можно было бы ожидать.

**Листинг 10.21. Извлечение информации о файле**

```
private void button3_Click(object sender, EventArgs e)
{
    System.Diagnostics.FileVersionInfo info =
        System.Diagnostics.FileVersionInfo.GetVersionInfo(
                                    @"C:\windows\system32\mspaint.exe");
```

```
    // Выводим информацию о выбранном файле
    listBox1.Items.Add("Выбранный файл: " + info.FileName);
    listBox1.Items.Add("Product Name: " + info.ProductName);
    listBox1.Items.Add("Product Version: " + info.ProductVersion);
    listBox1.Items.Add("Company Name: " + info.CompanyName);
    listBox1.Items.Add("File Version: " + info.FileVersion);
    listBox1.Items.Add("File Description: " + info.FileDescription);
    listBox1.Items.Add("Original Filename: " + info.OriginalFilename);
    listBox1.Items.Add("Legal Copyright: " + info.LegalCopyright);
    listBox1.Items.Add("InternalName: " + info.InternalName);
    listBox1.Items.Add("IsDebug: " + info.IsDebug);
    listBox1.Items.Add("IsPatched: " + info.IsPatched);
    listBox1.Items.Add("IsPreRelease: " + info.IsPreRelease);
    listBox1.Items.Add("IsPrivateBuild: " + info.IsPrivateBuild);
    listBox1.Items.Add("IsSpecialBuild: " + info.IsSpecialBuild);
}
```

## Как создать временный файл?

Для создания временного файла, который будет иметь уникальное имя и находиться в папке, определенной переменной окружения TEMP, нужно воспользоваться методом `GetTempFileName` класса `System.IO.Path` (листинг 10.22). Причем сама система позаботится об уникальности имени создаваемого файла.

**Листинг 10.22. Создание временного файла**

```
private void button5_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
    // Создаем временный файл
    listBox1.Items.Add(System.IO.Path.GetTempFileName());
}
```

## Как создать уникальное имя для файла?

В .NET Framework 2.0 появился новый метод `GetRandomFileName`, который относится к классу `System.IO.Path`. Метод `GetRandomFileName` возвращает случайным образом сгенерированную строку, которую можно использовать в качестве имени файла (листинг 10.23). Обратите внимание, что в отличие от метода `GetTempFileName`, метод `GetRandomFileName` не создает сам файл.

**Листинг 10.23. Создание уникального имени для файла**

```
private void button16_Click(object sender, EventArgs e)
{
     string randomFile = System.IO.Path.GetRandomFileName();
     MessageBox.Show(randomFile); // вернет что-то типа 5wvzx2n0.lby
}
```

При вызове данного метода получаются очень странные имена файлов типа 4rvzx0w.ls4. Если вам необходимо создать текстовый файл с уникальным именем, то удобнее использовать метод `GetRandomFileName` в связке с методом `ChangeExtension`, как показано в листинге 10.24.

**Листинг 10.24. Создание уникального текстового файла**

```
string tempText = System.IO.Path.ChangeExtension(
                      System.IO.Path.GetRandomFileName(), ".txt");
MessageBox.Show(tempText);
```

## Как ограничить доступ к файлу?

Если вам необходимо закрыть доступ к файлу, то вы можете установить желаемый уровень доступа через перечисление `FileShare`. Пример показан в листинге 10.25.

**Листинг 10.25. Ограничение доступа к файлу**

```
string fileName = @"c:\test.txt";
FileStream stream = new FileStream(fileName,
                        FileMode.Open, FileAccess.Read, FileShare.None);
//никому не даем доступ к файлу

// здесь находится ваш код
...
// открываем снова файл для доступа
stream.Close();
```

Существует еще один вариант ограничения доступа к файлу (листинг 10.26).

**Листинг 10.26. Второй способ ограничения доступа**

```
string fileName = @"c:\test.txt";
FileStream stream = File.Open(fileName, FileMode.Open);
stream.Lock(0, stream.Length); // блокируем файл

// здесь ваш код
...
// снимаем блокировку
stream.Unlock(0, stream.Length);
```

## Как работать с бинарными файлами?

Для чтения и записи бинарных файлов используются классы `BinaryReader` и `BinaryWriter` пространства имен `System.IO`. Пример приведен в листинге 10.27.

**Листинг 10.27. Работа с бинарными файлами**

```
private void button9_Click(object sender, EventArgs e)
{
    string fileName = @"c:\wutemp\test.bin";
    // Создадим новый пустой файл
    if (System.IO.File.Exists(fileName))
    {
        MessageBox.Show("Указанный файл уже существует!", fileName);
        return;
    }

    System.IO.FileStream fs = new System.IO.FileStream(fileName,
                                        System.IO.FileMode.CreateNew);

    // Запишем данные в файл
    System.IO.BinaryWriter w = new System.IO.BinaryWriter(fs);

    for (int i = 0; i < 11; i++)
    {
        w.Write((int)i);
    }
    w.Close();
    fs.Close();
```

```
    // Попытаемся прочитать записанное
    fs = new System.IO.FileStream(fileName, System.IO.FileMode.Open,
                                  System.IO.FileAccess.Read);
    System.IO.BinaryReader binread = new System.IO.BinaryReader(fs);

    // считываем данные из test.bin
    for (int i = 0; i < 11; i++)
    {
        MessageBox.Show(binread.ReadInt32().ToString());
    }
    w.Close();
}
```

В этом примере создается бинарный файл test.bin. Попробуйте открыть его в текстовом редакторе Блокнот, чтобы убедиться, что это действительно бинарный файл, а не стандартный текстовый файл.

## Как работать с текстовыми файлами?

Для записи и чтения текстовых файлов следует использовать классы `StreamWriter` и `StreamReader`. Простейшие приемы работы с ними показаны в листинге 10.28.

**Листинг 10.28. Чтение и запись текстовых файлов**

```
private void button10_Click(object sender, EventArgs e)
{
    string fileName = @"c:\wutemp\text.txt";

    if (System.IO.File.Exists(fileName))
    {
        MessageBox.Show("Указанный файл уже существует.", fileName);
        return;
    }

    System.IO.StreamWriter sr = System.IO.File.CreateText(fileName);
    sr.WriteLine("Раз, два, три, четыре, пять");
    sr.WriteLine("1, 2, 3. 9 1/2 и так далее");
    sr.WriteLine("Я изучаю {0} и {1}.", "C#", "Visual Basic");
    sr.Close();
}
```

## Как добавить текст в существующий файл?

Чтобы добавить некоторый текст в конец уже существующего текстового файла, нужно воспользоваться методом `AppendText` класса `StreamWriter`. Давайте попытаемся добавить еще одну строчку в файл, который мы создали только что (листинг 10.29).

**Листинг 10.29. Добавление текста в существующий файл**

```
private void button11_Click(object sender, EventArgs e)
{
    string fileName = @"c:\wutemp\text.txt";

    // Добавляем одну строчку в текстовый файл
    using (System.IO.StreamWriter sw =
                            System.IO.File.AppendText(fileName))
    {
        sw.WriteLine("Добавили еще одну строчку");
    }

    // Прочитаем текст из файла
    using (System.IO.StreamReader sr = System.IO.File.OpenText(fileName))
    {
        string s = "";
        while ((s = sr.ReadLine()) != null)
        {
            MessageBox.Show(s);
        }
    }
}
```

## Построчное чтение текстового файла

Построчное чтение текстового файла применяется, например, для загрузки текста в список `ListBox`. С помощью метода `ReadLine` класса `System.IO.StreamReader` мы считываем текст строчка за строчкой (листинг 10.30). Здесь особое внимание нужно обращать на момент достижения конца файла, чтобы избежать ошибок. В предыдущем примере мы уже прочитали текст из файла построчно. Вот еще один похожий простой способ для разнообразия.

**Листинг 10.30. Построчное чтение текстового файла**

```
private void button12_Click(object sender, EventArgs e)
{
    // Создадим экземпляр StreamReader для чтения из файла
    using (System.IO.StreamReader sr =
                            new IO.StreamReader(@"c:\wutemp\text.txt"))
    {
        String line;
        // Читаем каждую строку, пока не достигнем конца файла
        while ((line = sr.ReadLine()) != null)
        {
            MessageBox.Show(line);
        }
    }
}
```

## Загрузить текстовый файл в список?

Надо сказать, что в .NET Framework 2.0 появились новые методы `ReadAllLines` и `ReadAllText`, которые также позволяют считывать данные из текстового файла. Поэтому, чтобы не отставать от времени, в новых проектах будем использовать более современные приемы программирования (листинг 10.31).

**Листинг 10.31. Загрузка текстового файла в список**

```
private void button13_Click(object sender, EventArgs e)
{
    int index = 2;
    //номер строки, с которой начинать
    string path = @"c:\wutemp\text.txt";  //путь к файлу
    string[] allLines = System.IO.File.ReadAllLines(path);
    for (int i = index; i < allLines.Length - 1; i++)
    {
        listBox1.Items.Add(allLines[i]);
    }
}
```

Обратите внимание, как легко стало обращаться со строками. Например, мы можем использовать значение индекса, чтобы читать файл не с самого начала, а с третьей строчки, и читать текст до предпоследней строки.

### *Примечание*

Кроме описанных методов `ReadAllLines` и `ReadAllText` в .NET Framework 2.0 появилось еще несколько удобных методов для работы с файлами. Обязательно почитайте документацию.

Давайте остановимся на этих примерах, чтобы лучше разобраться в новых методах. До появления метода `ReadAllText` для получения данных из текстового файла нужно было писать код вроде приведенного в листинге 10.32.

**Листинг 10.32. Чтение данных из текстового файла (старый способ)**

```
StringBuilder sb = new StringBuilder();
using (StreamReader sr = new StreamReader(filename))
{
    while (sr.Peek() >= 0)
    {
        sb.AppendLine(sr.ReadLine());
    }
}
textBox1.Text = sb.ToString();
```

В этом примере мы использовали метод класса `StreamReader`. Сравните этот код с кодом в листинге 10.33, где используется новый метод `ReadAllText`.

**Листинг 10.33. Чтение данных из текстового файла (новый способ)**

```
// Получение данных из файла с использованием метода ReadAllText
textBox1.Text = File.ReadAllText(filename);
```

Например, если нужно просто открыть какой-то текстовый файл и скопировать его содержимое в текстовое поле, то метод `ReadAllText` идеально подходит для этой задачи. Для метода `ReadAllText` нет необходимости заключать вызов в блок `try`, поскольку прикладной программный интерфейс сам выполнит важную задачу закрытия файла. Производительность в обоих случаях примерно одинаковая, но использовать метод `ReadAllText` гораздо проще. Новые интерфейсы прикладного программирования для класса `File` (`ReadAllLines`, `ReadAllText`, `ReadAllBytes`) считывают весь файл сразу и сохраняют эту информацию в памяти. Если при извлечении данных нужно выполнить преобразование в соответствии с некоторой логикой или если данные преобразуются в другие типы, то, возможно, лучше считывать по одной

строке за раз и освобождать данные при переходе к следующей строке. Особенно это существенно при работе с большими файлами. При использовании метода `File.ReadAllLines` данные из файла будут все время находиться в памяти, а при использовании метода `StreamReader.ReadLine` будет загружена только одна строка.

## Как получить короткое имя файла из длинного файла и наоборот?

Ветераны программирования, работавшие с операционной системой DOS, помнят те времена, когда было ограничение "8.3" на число символов в имени файла и расширении. И, например, такая популярная в свое время программа, как Norton Commander, автоматически создавала из длинных имен короткое по схеме "8.3". В современном программировании необходимость преобразовывать длинное имя файла в короткое уходит в прошлое, но на всякий случай возьмите этот пример (листинг 10.34) на заметку. Для преобразования имен нам понадобятся две функции Windows API `GetShortPathName` и `GetLongPathName`.

**Листинг 10.34. Получение короткого и длинного имени файла**

```
[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
public static extern int GetShortPathName(
[MarshalAs(UnmanagedType.LPTStr)] string path,
[MarshalAs(UnmanagedType.LPTStr)] StringBuilder shortPath,
int shortPathLength
);

[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
public static extern int GetLongPathName(
[MarshalAs(UnmanagedType.LPTStr)] string path,
[MarshalAs(UnmanagedType.LPTStr)] StringBuilder longPath,
int longPathLength
);

private void button14_Click(object sender, EventArgs e)
{
    StringBuilder shortPath = new StringBuilder(1024);
    GetShortPathName(@"C:\Program Files\Microsoft Silverlight\" +
                     @"Microsoft.Scripting.Silverlight.dll",
```

```
                    shortPath, shortPath.Capacity);
    this.Text = shortPath.ToString();

     StringBuilder longPath = new StringBuilder(1024);
     GetLongPathName(this.Text, longPath, longPath.Capacity);
     listBox1.Items.Clear();
     listBox1.Items.Add(longPath.ToString());
 }
```

## Как удалить файл в Корзину

Если вам нужно удалить файл в Корзину, как это принято в Windows, то придется вызывать функцию Windows API `SHFileOperation`. В листинге 10.35 в корзину удаляется файл test.txt, который находится на диске C:\.

**Листинг 10.35. Удаление файла в корзину**

```
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto, Pack = 1)]
private struct SHFILEOPSTRUCT
{
    public IntPtr hwnd;
    [MarshalAs(UnmanagedType.U4)]
    public int wFunc;
    public string pFrom;
    public string pTo;
    public short fFlags;
    [MarshalAs(UnmanagedType.Bool)]
    public bool fAnyOperationsAborted;
    public IntPtr hNameMappings;
    public string lpszProgressTitle;
 }

[DllImport("shell32.dll", CharSet = CharSet.Auto)]
private static extern int SHFileOperation(ref SHFILEOPSTRUCT fileOp);

// константы для SHFileOperation
private const int FO_DELETE = 3;
private const int FOF_ALLOWUNDO = 64;
private const int FOF_NOCONFIRMATION = 16;
```

```
public static int MoveToRecycleBin(string path)
{
    SHFILEOPSTRUCT fop = new SHFILEOPSTRUCT();
    fop.wFunc = FO_DELETE;
    fop.pFrom = path + "\0"; // без дополнительного \0 не работает
    fop.fFlags = FOF_ALLOWUNDO | FOF_NOCONFIRMATION;

    return SHFileOperation(ref fop);
}

private void button5_Click(object sender, EventArgs e)
{
    // Удаляем файл в корзину
    MoveToRecycleBin(@"C:\Test.txt");
}
```

По старой привычке решил посмотреть информацию на эту тему в Интернете, и меня ждала удача. В блоге под названием TK'S Blog (**http://blogs.gotdotnet.ru/personal/tk/default.aspx**) обнаружил заметку, в которой говорилось, как можно удалить файл без вызова неуправляемого кода. Для этого нужно подключить сборку `Microsoft.VisualBasic` (через **Project | Add Reference…**) и далее написать код, приведенный в листинге 10.36.

**Листинг 10.36. Удаление файла через управляемый код**

```
private void butDelRecycleVB_Click(object sender, EventArgs e)
{
    FileSystem.DeleteFile("c:/temp.txt",
                  UIOption.AllDialogs,RecycleOption.SendToRecycleBin);
}
```

Обратите внимание, что метод `DeleleFile` является новым методом, который появился в .NET Framework 2.0. Для удаления папок в Корзину можете применить также новый метод `FileSystem.DeleteDirectory`, который работает схожим образом.

> ***Примечание***
>
> Примеры работы с файлами находятся в папке FilesDemo на прилагаемом диске.

## Как записать и прочитать текст в различных кодировках?

Как правило, при чтении или записи английских текстов никаких проблем не возникает. А вот при чтении русских текстов иногда возникают трудности. Дело в том, что так исторически сложилось, что имеется несколько кодировок для русского языка. В настоящее время самой распространенной кодировкой является Windows-1251, но иногда попадаются текстовые файлы, записанные в кодировке KOI8-R или DOS. В этом случае нужно использовать соответствующую кодировку при помощи метода `GetEncoding` класса `Encoding`. Пример показан в листинге 10.37.

**Листинг 10.37. Чтение и запись в различных кодировках**

```
private void butKOI8R_Click(object sender, EventArgs e)
{
    System.IO.FileStream fs =
        new O.FileStream(@"D:\koi8r.txt",
                         System.IO.FileMode.OpenOrCreate);
    System.IO.StreamReader sr =
        new System.IO.StreamReader(fs,
                         System.Text.Encoding.GetEncoding("KOI8-r"),
                         false);
    textBox1.Text = sr.ReadToEnd();
}

private void butSaveWin1251_Click(object sender, EventArgs e)
{
    System.IO.StreamWriter sw =
        new System.IO.StreamWriter(@"D:\MyBook\CSharp_Tips\win.txt",
                                   false,
                                   ext.Encoding.GetEncoding(1251));
    sw.Write(textBox1.Text);
    sw.Close();
}
```

Сначала мы загружаем текст из файла koi8r.txt, который записан в кодировке KOI8-R, поэтому в коде явно указываем имя кодировки. Далее мы сохраняем текст из текстового поля в файл уже в кодировке Windows-1251, используя вместо имени кодировки целочисленное значение 1251. Не забудьте при использовании примеров исправить пути к текстовым файлам. В папке Encod-

ing лежат готовые файлы koi8-r.txt и win.txt, записанные в соответствующих кодировках, которые вы можете использовать для проверки работы кода.

***ПРИМЕЧАНИЕ***

Пример находится в папке Encoding на прилагаемом компакт-диске.

## Как прочитать XML-файлы?

Кроме обычных текстовых файлов, у программиста возникает потребность открыть для чтения XML-файлы. Это становится все более актуальной задачей в связи с растущей популярностью формата XML. На сайте Microsoft имеется статья из базы знаний "Чтение инструкций XML из файла с помощью Visual C# 2005 или Visual C# .NET", которая находится по адресу **http://support.microsoft.com/kb/307548/ru**. Это один из редких случаев, когда статья дана на русском языке. Причем статья неоднократно редактировалась (текущая версия статьи 3.4). Вкратце расскажу самое основное из этой статьи. В статье описывается использование класса `XmlTextReader`, который предоставляет возможность выполнять прямой синтаксический разбор и выделение инструкций языка XML (eXtensible Markup Language). В статье приводится быстрый потоковый доступ к XML с использованием маркеров, являющийся альтернативой использованию объектной модели, такой как XML DOM (Document Object Model). Для примера я создал XML-файл bhvbooks.xml (листинг 10.38), в котором содержится информация о некоторых книгах издательства "БХВ-Петербург".

**Листинг 10.38. Содержание файла bhvbooks.xml**

```
<?xml version='1.0' encoding="windows-1251"?>
<!-- Books from BHV -->
<bookstore>
  <book genre="internet" publicationdate="2007" ISBN="0000">
    <title>Электронная почта. Народные советы.</title>
    <author>
      <first-name>П.</first-name>
      <last-name>Данилов</last-name>
    </author>
    <price>269.00</price>
  </book>
  <book genre="video" publicationdate="2007" ISBN="0001">
```

```
    <title>Цифровое видео. Народные советы (+Видеокурс на CD)</title>
    <author>
      <first-name>В.П.</first-name>
      <last-name>Молочков</last-name>
    </author>
    <price>254.00</price>
  </book>
  <book genre="security" publicationdate="2007" ISBN="0003">
    <title>Безопасность  домашнего ПК. Народные советы.</title>
    <author>
      <name>Крупник А.</name>
    </author>
    <price>194.00</price>
  </book>
</bookstore>
```

Файл bhvbooks.xml необходимо скопировать в папку \Bin\Debug, которая расположена в папке, где создается данный проект. Создайте новый проект и подключите пространство имен `System.Xml` при помощи директивы `using`, чтобы избавиться от необходимости указывать объявления `XmlTextReader` в коде.

```
using System.Xml;
```

Класс `XmlTextReader` обеспечивает быстрый механизм чтения XML. Нам нужно создать экземпляр объекта `XmlTextReader` и загрузить файл bhvbooks.xml. Добавьте приведенный ниже код в конструктор формы.

```
XmlTextReader reader = new XmlTextReader ("bhvbooks.xml");
```

Теперь мы можем прочитать данные XML-файл при помощи метода `Read`. Метод `Read` последовательно перемещается по XML-файлу до достижения конца файла, после чего возвращает значение `false`. (Отметим, что в листинге 10.39 показан внешний цикл `while`, а в следующих двух действиях показано использование этого цикла для чтения XML).

**Листинг 10.39. Использование цикла `while` для чтения XML-файла**

```
while (reader.Read())
{
    // Обработка данных.
    listBox1.Items.Add("<" + reader.Name + ">");
}
Console.ReadLine();
```

Для обработки данных XML в каждой записи содержится узел, тип которого можно определить по свойству `NodeType`. Свойства `Name` и `Value` возвращают имя узла (имена элемента и атрибута) и значение узла (текст узла) текущего узла (или записи). Перечисление `NodeType` определяет тип узла. В листинге 10.40 приведен код, в котором выводятся имена элементов и типы документа. Имейте в виду, что в данном примере пропущены атрибуты элементов.

**Листинг 10.40. Чтение имен элементов и типов документа**

```
while (reader.Read())
{
    switch (reader.NodeType)
    {
        case XmlNodeType.Element: // Узел является элементом.
            listBox1.Items.Add("<" + reader.Name + ">");
            break;
        case XmlNodeType.Text: // Вывести текст в каждом элементе.
            listBox1.Items.Add(reader.Value);
            break;
        case XmlNodeType.EndElement: // Вывести конец элемента.
            listBox1.Items.Add("</" + reader.Name + ">");
            break;
    }
}
```

Типы узлов элементов могут включать списки связанных с ними узлов атрибутов. Метод `MovetoNextAttribute` последовательно перемещается по всем атрибутам элемента. Свойство `HasAttributes` используется для проверки наличия у узла атрибутов. Свойство `AttributeCount` возвращает количество атрибутов для текущего узла. Использование метода `MovetoNextAttribute` показано в листинге 10.41, в котором представлен полный код, разбирающий XML-файл.

**Листинг 10.41. Полный листинг кода**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
```

```
using System.Windows.Forms;
using System.Xml;


namespace XMLTextReaderDemo
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            XmlTextReader reader = new XmlTextReader("bhvbooks.xml");
            while (reader.Read())
            {
                switch (reader.NodeType)
                {
                    case XmlNodeType.Element: // Узел является элементом.
                        listBox1.Items.Add("<" + reader.Name);

                        while (reader.MoveToNextAttribute())
                            // Чтение атрибутов.
                            listBox1.Items.Add(" " + reader.Name +
                                "='" + reader.Value + "'");
                        listBox1.Items.Add(">");
                        break;
                    case XmlNodeType.Text:
                        // Вывести текст в каждом элементе.
                        listBox1.Items.Add(reader.Value);
                        break;
                    case XmlNodeType.EndElement:
                        // Вывести конец элемента.
                        listBox1.Items.Add("</" + reader.Name + ">");
                        break;
                }
            }
        }
    }
}
```

### *Примечание*

Пример находится в папке XMLTextReaderDemo на прилагаемом диске.

## Сравнение двух файлов

Если вам необходимо сравнить содержимое двух файлов, то придется сравнивать байт за байтом оба файла. И опять нам на помощь придет статья "How to create a File-Compare function in Visual C#" из Базы Знаний Microsoft, которую можно найти по адресу **http://support.microsoft.com/kb/320348/**. Обратите внимание, что мы будем сравнивать именно содержимое файлов, а не их имена, даты создания и другие атрибуты. Кстати, существует такая утилита командной строки fc.exe, входящая в состав операционных систем Windows, MS-DOS и некоторых средств разработки, которая выполняет такую же работу. Принцип работы приложения не сложен (листинг 10.42). Программа сравнивает байт за байтом два файла, пока не будет достигнут конец файла. Чтобы не делать лишнюю работу, программа использует две проверки, которые повышают эффективность ее работы. Во-первых, если сравнивается один и тот же файл (указаны одинаковые пути к файлам), то побайтное сравнение не производится, так как и так понятно, что содержимое одинаково. Во-вторых, если размер файлов различается, то проверка по байтам также не производится, так как файлы разной длины не могут быть одинаковыми. Для работы нам понадобятся два текстовых поля, в которых пользователь должен ввести путь к сравниваемым файлам, и кнопка.

**Листинг 10.42. Сравнение двух файлов**

```
using System.IO;

// Метод для сравнения файлов
// В качестве параметров используются две строки,
// содержащие полные пути к сравниваемым файлам.
// Возвращается не 0, если содержимое файлов одинаково.
// Если содержимое файлов не одинаково, то возвращается 0.
private bool FileCompare(string file1, string file2)
{
    int file1byte;
    int file2byte;
    FileStream fs1;
    FileStream fs2;

    // Если пути файлов совпадают
    if (file1 == file2)
    {
```

```
        // Возвращаем true, так как сравнивается один и тот же файл.
        return true;
    }

    // Открываем два файла.
    fs1 = new FileStream(file1, FileMode.Open);
    fs2 = new FileStream(file2, FileMode.Open);

    // Проверяем размеры файла. Если размеры отличаются,
    // то файлы не одинаковы.
    if (fs1.Length != fs2.Length)
    {
        // Закрываем файлы
        fs1.Close();
        fs2.Close();

        // Возвращаем false, так как файлы не одинаковы
        return false;
    }

    // Читаем и сравниваем байт каждого файла
    // пока не встретим несовпадающий байт
    // или пока не будет достигнут конец файла.
    do
    {
        // Считываем один байт из каждого файла.
        file1byte = fs1.ReadByte();
        file2byte = fs2.ReadByte();
    }
    while ((file1byte == file2byte) && (file1byte != -1));

    // Закрываем файлы.
    fs1.Close();
    fs2.Close();

    // Возвращается true в случае успешного сравнения всех байтов.
    // file1byte равен file2byte, если содержимое файлов
    // полностью совпадает.
    return ((file1byte - file2byte) == 0);
}
```

```
private void button1_Click(object sender, EventArgs e)
{
    // Сравниваем два файла. Путь к файлам указан в текстовых полях.
    if (FileCompare(this.textBox1.Text, this.textBox2.Text))
    {
        MessageBox.Show("Файлы одинаковы.");
    }
    else
    {
        MessageBox.Show("Файлы не одинаковы.");
    }
}
```

***Примечание***

Пример сравнения двух файлов находится в папке FileCompare на прилагаемом диске.

## Отслеживание изменений в файловой системе

В .NET Framework имеется такой интересный класс, как `FileSystemWatcher` из пространства имен `System.IO`, который позволяет следить за изменениями в файловой системе. Например, вы можете установить наблюдение за заданной папкой и отслеживать любые изменения в подпапках и файлах. Обратите внимание, что наблюдать за файлами можно не только на локальном компьютере, но и на сетевых дисках или удаленных компьютерах. Я приведу лишь базовый пример (листинг 10.43), который даст представление о классе `FilySystemWatcher`.

**Листинг 10.43. Отслеживание изменений в файловой системе**

```
private void Form1_Load(object sender, EventArgs e)
{
    // Создадим объект FileSystemWatcher и установим его свойства
    FileSystemWatcher watcher = new FileSystemWatcher();
    // Наблюдение за папкой WUTEMP
    watcher.Path = @"C:\wutemp\";
    // Следим за изменениями по последнему доступу,
    // времени записи и переименования файлов или папки
```

```
    watcher.NotifyFilter = NotifyFilters.LastAccess
        | NotifyFilters.LastWrite
        | NotifyFilters.FileName | NotifyFilters.DirectoryName;
    // Наблюдаем только файлы TXT
    watcher.Filter = "*.txt";

    // Добавляем обработчики событий
    watcher.Changed += new FileSystemEventHandler(OnChanged);
    watcher.Created += new FileSystemEventHandler(OnChanged);
    watcher.Deleted += new FileSystemEventHandler(OnChanged);
    watcher.Renamed += new RenamedEventHandler(OnRenamed);

    // Начинаем наблюдение
    watcher.EnableRaisingEvents = true;
}

// События
private static void OnChanged(object source, FileSystemEventArgs e)
{
    // При изменении, создании или удалении файла выводим сообщение
    MessageBox.Show("Файл: " +  e.FullPath + " " + e.ChangeType);
}

private static void OnRenamed(object source, RenamedEventArgs e)
{
    // При переименовании файла тоже выводим сообщение
    MessageBox.Show("Файл: " + e.OldFullPath +
                    " переименован в " + e.FullPath);
}
```

Можно написать приложение, которое при изменениях в наблюдаемых папках будет отсылать сообщения по электронной почте. Учитывая, что слежение за папками можно проводить и на удаленных машинах, вы можете использовать эту возможность для разработки полезных утилит, которые пригодятся системному администратору.

### *Примечание*

Пример работы с классом `FileSystemWatcher` находится в папке Watcher на прилагаемом диске.

## Как установить уровень доступа к файлу?

Закончить главу я хочу кратким обзором новой возможности настройки уровня доступа к файлам и папками. В .NET Framework 2.0 появились новые методы `GetAccessControl` и `SetAccessControl` классов `File` и `Directory`, с помощью которых можно прочитать или установить уровень доступа к файлу или папке. Общая методика изменения уровня доступа к файлу следующая. Сначала вы выбираете файл, настройки доступа которого вы хотите изменить, далее с помощью метода `GetAccessControl` узнаете текущие настройки безопасности, и затем при помощи `SetAccessControl` устанавливаете уже нужные настройки для доступа. В следующем примере я устанавливаю настройки к файлу test.txt для пользователя alexander (то есть для себя), в результате чего при попытке открыть файл для чтения я получу сообщение, что мне отказано в доступе. Чтобы вернуть доступ к файлу, нужно снова изменить настройки.

**Листинг 10.44. Установка уровня доступа к файлу**

```
using System.Security.AccessControl;
using System.IO;

static void SetRule(string filePath, string account,
    FileSystemRights rights, AccessControlType controlType)
{
    // Получим объект FileSecurity, который устанавливает
    // текущие настройки безопасности
    FileSecurity fSecurity = File.GetAccessControl(filePath);
    // Добавим FileSystemAccessRule к настройкам безопасности
    fSecurity.ResetAccessRule(new FileSystemAccessRule(account,
                                                rights, controlType));
    // Устанавливаем новые настройки доступа
    File.SetAccessControl(filePath, fSecurity);
}

private void butDenyAccess_Click(object sender, EventArgs e)
{
    // Отказать в доступе пользователю alexander
    SetRule(@"d:\test.txt", "alexander",
            FileSystemRights.Read, AccessControlType.Deny);
}
```

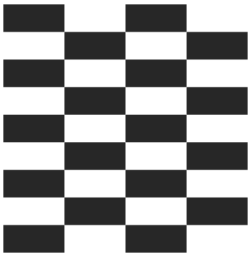
```
private void butAllowAccess_Click(object sender, EventArgs e)
{
    // Разрешить доступ к файлу пользователю alexander
    SetRule(@"d:\test.txt", "alexander",
            FileSystemRights.Read, AccessControlType.Allow);
}
```

При попытке открыть файл после нажатия на первую кнопку я получаю сообщение "Отказано в доступе". Если вас заинтересовала данная тема, то начните изучать документацию по этому вопросу. Компания Microsoft в последнее время уделяет много времени безопасности выпускаемых продуктов и призывает разработчиков также использовать новые возможности по настройке уровней безопасности в своих программах.

> ***Примечание***
>
> Пример контроля за доступом к файлам находится в папке AccessFile на прилагаемом диске.

# Заключение

Работа с файловой системой — одна из наиболее распространенных задач, возникающих в работе программиста. Поэтому знание основных операций с дисками, папками и файлами просто необходимо любому разработчику. Надеюсь, приведенные в этой главе советы помогут вам в разработке собственных программ.



# Глава 11

# Библиотека WSH

Многие системные администраторы пишут сценарии с использованием технологии WSH. С помощью WSH легко выполнять многие операции в системе. Разработчики, программирующие на C#, могут использовать классы библиотеки `IWshRuntimeLibrary`, которая является оболочкой для технологии WSH. Чтобы подключить данную библиотеку к проекту, необходимо подключить нужный объект Windows Script Host Object Model. Для этого в среде разработки выбираем в окне, вызываемом меню **Project | Add Reference…**, вкладку **COM** и ищем строчку **Windows Script Host Object Model**. Сам объект находится в файле wshom.ocx. Теперь необходимо в начале проекта добавить строку

```
using IWshRuntimeLibrary;
```

После этого вы можете использовать все классы этой библиотеки в своих приложениях. Рассмотрим несколько стандартных задач, которые легко решаются с использованием WSH.

## Создание ярлыка

Создать ярлык с помощью библиотеки `IWshRuntimeLibrary` очень просто, используя встроенные возможности WSH. Пример показан в листинге 11.1.

**Листинг 11.1. Создание ярлыка**

```
private void butCreateShortcut_Click(object sender, EventArgs e)
{
    // Создадим ярлык на Рабочем столе
    object shortDesktop = (object)"Desktop";
```

```
    WshShell shell = new WshShell();
    // Путь к ярлыку
    string shortcutAddress =
        (string)shell.SpecialFolders.Item(ref shortDesktop) +
         @"\Блокнотик.lnk";

    // Создаем объект ярлыка
    IWshShortcut shortcut =
        (IWshShortcut)shell.CreateShortcut(shortcutAddress);

    // Задаем свойства для ярлыка
    // Описание ярлыка в всплывающей подсказке
    shortcut.Description = "Ярлык для текстового редактора";
    // Горячая клавиша
    shortcut.Hotkey = "Ctrl+Shift+N";
    // Путь к самой программе Блокнот
    shortcut.TargetPath =
        Environment.GetFolderPath(Environment.SpecialFolder.System) +
        @"\notepad.exe";

    // Все готово. Можно создавать ярлык
    shortcut.Save();
}
```

В переменной `shortDesktop` мы использовали объект `Desktop`. Также можно использовать и другие объекты, которые очень похожи на перечисление `SpecialFolder` из библиотеки классов .NET Framework, о котором шла речь в *главе 10*. Например, имеются такие объекты, как `Templates`, `Favorites`, `Recent`, `Startup` и т. д.

# Получение списка установленных в системе принтеров

С помощью библиотеки WSH также легко можно получить список установленных в системе принтеров. В листинге 11.2 показано, как это сделать.

**Листинг 11.2. Получение списка установленных принтеров**

```
using System.Collections;
using IWshRuntimeLibrary;
```

```
private void butGetPrinters_Click(object sender, EventArgs e)
{
    WshNetwork network = new WshNetwork();
    // Получим список всех принтеров
    foreach (IEnumerable printer in network.EnumPrinterConnections())
    {
        listBox1.Items.Add(printer);
    }
}
```

# Установка принтера по умолчанию

Если у вас в системе установлено несколько принтеров, то с помощью несложного кода (листинг 11.3) можно установить принтер, используемый по умолчанию.

**Листинг 11.3. Установка принтера по умолчанию**

```
using IWshRuntimeLibrary;

private void butSetDefaultPrinter_Click(object sender, EventArgs e)
{
    WshNetwork network = new WshNetwork();

    // Получим список принтеров
    IWshCollection Printers = network.EnumPrinterConnections();

    if (Printers.Count() > 0)
    {
        // Выбираем индекс устанавливаемого принтера
        object index = (object)"1";
        // Устанавливаем выбранный принтер как принтер по умолчанию
        network.SetDefaultPrinter((string)Printers.Item(ref index));
    }
}
```

# Получение списка сетевых дисков

Также библиотека WSH позволяет быстро получить список сетевых дисков, как показано в листинге 11.4.

**Листинг 11.4. Получение списка сетевых дисков**

```
using System.Collections;
using IWshRuntimeLibrary;

private void butNetworkDrives_Click(object sender, EventArgs e)
{
    WshNetwork network = new WshNetwork();
    // Перебираем все сетевые диски
    foreach (IEnumerable driver in network.EnumNetworkDrives())
    {
        MessageBox.Show(driver.ToString());
    }
}
```

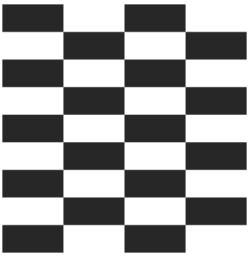
Библиотека WSH позволяет также работать с дисками, папками, файлами, реестром. Поэтому если в вашем арсенале имеются готовые сценарии, написанные на языке WSH, то вы без труда сможете использовать эти наработки в новых проектах.

> ***Примечание***
>
> Примеры работы с библиотекой WSH находятся в папке WSHDemo на прилагаемом диске.

# Заключение

Использование библиотеки WSH пригодится тем, кто привык применять эту технологию в сценариях. В этом случае не нужно переучиваться и изучать, как другим способом можно реализовать уже знакомые возможности, предоставляемые этой технологией. Хотя, с другой стороны, многие действия, которые возможно осуществить, используя WSH, можно осуществить и с помощью стандартных классов .NET Framework.

# Глава 12



# WMI

Прежде чем мы рассмотрим примеры работы с WMI, необходимо сказать несколько слов о самой технологии WMI. WMI (Windows Management Instrumentation) — это специально разработанный компанией Microsoft интерфейс управления Windows, основанный на определенных стандартах. Технология WMI широко используется системными администраторами при помощи специально написанных сценариев. Но WMI можно использовать и в приложениях C#. WMI включена во все современные версии Windows, а также ее можно бесплатно загрузить с сайта Microsoft. Мы с вами рассмотрим множество примеров, основанных на этой универсальной технологии. Классы, предназначенные для работы с WMI, находятся в пространстве имен `System.Management`. Вы увидите, что с помощью WMI можно узнать множество информации о компьютерах. Часть этой информации мы уже получали при помощи встроенных средств .NET Framework или функций Windows API. Но эту информацию мы получали лишь для локальной машины. А технология WMI позволяет работать с удаленными компьютерами, и именно эта возможность представляет особый интерес для программистов.

## Использование WMI на удаленной машине

Прежде чем получить нужную информацию с удаленного компьютера, к нему нужно подключиться. В листинге 12.1 приведен небольшой кусок кода, который выполняет эту задачу.

**Листинг 12.1. Использование WMI на удаленной машине**

```
private void button1_Click(object sender, EventArgs e)
{
    ConnectionOptions options = new ConnectionOptions();
```

```
    // введите домен и учетную запись
    options.Username = @"mgu\administrator";
    // введите пароль
    options.Password = "12345678";
    // Укажите имя машины вместо buhgalter
    ManagementScope scope =
        new ManagementScope(@"\\buhgalter\root\cimv2", options);
    scope.Connect();
    MessageBox.Show("Соединение установлено");
}
```

Для использования этого примера вам нужно указать реальное имя домена и учетную запись, которая имеет права администратора. Также вам нужно ввести реальный пароль, используемый учетной записью. Естественно, необходимо использовать реальное имя машины в вашей сети. В нашем примере это машина под именем buhgalter. Теперь, после соединения с удаленной машиной, вы можете писать код, извлекающий нужную информацию. Если же у вас нет возможности проверять примеры, которые будут описаны ниже, на удаленных компьютерах, то вы можете протестировать их на своей локальной машине. В этом случае код станет чуть проще (листинг 12.2).

**Листинг 12.2. Использование WMI на локальной машине**

```
ManagementScope scope =
    new ManagementScope("\\\\localhost\\root\\cimv2");
scope.Connect();
```

Как видите, для работы с локальным компьютером нет нужды указывать учетную запись и пароль, а также в качестве имени машины достаточно указать localhost.

# Информация об операционной системе

С помощью пространства имен `Win32_OperatingSystem` можно узнать свойства операционной системы — название системы, версию системы, путь к системной папке, имя машины, имя изготовителя системы и многое другое. Пример приведен в листинге 12.3.

**Листинг 12.3. Получение информации об операционной системе**

```
private void button2_Click(object sender, EventArgs e)
{
    // Соединяемся с удаленной машиной
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"domen\admin";
    options.Password = "pass";

    ManagementScope scope =
        new ManagementScope("\\\\myserver\\root\\cimv2", options);
    scope.Connect();

    // Запрашиваем информацию об операционной системе
    ObjectQuery query = new ObjectQuery(
        "SELECT * FROM Win32_OperatingSystem");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject m in queryCollection)
    {
        // Выводим информацию в текстовое поле
        string nl = Environment.NewLine;

        textBox1.Text = "Имя машины : " + m["CSName"] + nl;
        textBox1.Text += "Операционная система: " + m["Caption"] + nl;
        textBox1.Text += "Версия ОС: " + m["Version"] + nl;
        textBox1.Text += "Язык операционной системы: "
                            + m["OSLanguage"] + nl;
        textBox1.Text += "Зарегистрированный пользователь: "
                            + m["RegisteredUser"] + nl;
        textBox1.Text += "Серийный номер продукта: "
                            + m["SerialNumber"] + nl;
        textBox1.Text += "Время установки: "
                            + ManagementDateTimeConverter.ToDateTime(
                                    m["InstallDate"].ToString()) + nl;
        textBox1.Text += "Папка Windows: " + m["WindowsDirectory"] + nl;
        textBox1.Text += "Системная папка: " + m["SystemDirectory"] + nl;
        textBox1.Text += "Производитель: " + m["Manufacturer"] + nl;
        textBox1.Text += "Доступная физическая память: "
                            + m["FreePhysicalMemory"] + nl;
```

```
        textBox1.Text += "Текущее число процессов: "
                              + m["NumberOfProcesses"] + nl;
    }
}
```

Хотелось бы обратить ваше внимание, что в Windows Vista появилось новое свойство для класса `Win32_OperatingSystem`: `OperatingSystemSKU`, который содержит номер от 0 до 25. С помощью этого свойства можно определить редакцию Windows Vista. Вот неполный список соответствий возвращаемого номера и названия редакции операционной системы:

- 0 — не определено;
- 1 — Ultimate Edition;
- 2 — Home Basic Edition;
- 3 — Home Basic Premium Edition;
- 4 — Enterprise Edition;
- 5 — Home Basic N Edition.

# Информация о компьютере

При помощи пространства имен `Win32_ComputerSystem` можно узнать, какому домену принадлежит машина, имя изготовителя и модель компьютера. Как это сделать, показано в листинге 12.4.

**Листинг 12.4. Получение информации о компьютере**

```
private void butComputer_Click(object sender, EventArgs e)
{
    ConnectionOptions options = new ConnectionOptions();
    // Используйте ваши реальные учетные записи и пароли
    options.Username = @"gamma\admin";
    options.Password = "mypass";

    ManagementScope scope =
       new ManagementScope("\\\\boss\\root\\cimv2", options);
    scope.Connect();

    // Запрашиваем информацию о компьютере
    ObjectQuery query = new ObjectQuery(
```

```
        "SELECT * FROM Win32_ComputerSystem");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject m in queryCollection)
    {
        // Отображаем информацию о компьютере

        string nl = Environment.NewLine;

        textBox1.Text = "Домен : " + m["Domain"] + nl;
        textBox1.Text += "Изготовитель : " + m["Manufacturer"] + nl;
        textBox1.Text += "Модель: " + m["Model"] + nl;
    }
}
```

В этом случае также имеется новое свойство `PCSystemType`, доступное в Windows Vista. Это свойство позволяет определять тип компьютера. Вот несколько возможных значений данного свойства:

- 0 — не определено;
- 1 — настольный компьютер;
- 2 — мобильный компьютер (ноутбук);
- 3 — рабочая станция.

Кроме того, к уже имеющемуся свойству `NumberOfProcessors` в Windows Vista добавилось новое свойство `NumberOfLogicalProcessors`, которое позволяет получить число логических процессоров.

# Информация о производителе

С помощью пространства имен `Win32_ComputerSystemProduct` можно узнать чуть больше о производителе компьютера. В листинге 12.5 показано, как это можно сделать.

**Листинг 12.5. Получение информации о производителе**

```
private void butProduct_Click(object sender, EventArgs e)
{
```

```
    // Соединяемся с удаленной машиной
  ConnectionOptions options = new ConnectionOptions();
  options.Username = @"gamma\admin";
  options.Password = "mypass";

  ManagementScope scope =
     new ManagementScope("\\\\srv-sql\\root\\cimv2", options);
  scope.Connect();

    ObjectQuery query = new ObjectQuery(
        "SELECT * FROM Win32_ComputerSystemProduct");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject m in queryCollection)
    {
        // Отображаем информацию об удаленном компьютере

        string nl = Environment.NewLine;

        textBox1.Text = "Описание : " + m["Description"] + nl;
        textBox1.Text += "Серийный номер : "
            + m["IdentifyingNumber"] + nl;
        textBox1.Text += "Имя : " + m["Name"] + nl;
        textBox1.Text += "Идентификатор продукта : " + m["UUID"] + nl;
        textBox1.Text += "Производитель : " + m["Vendor"] + nl;
    }
}
```

# Получение информации о процессорах

Настало время получить информацию о процессорах. Для этого необходимо сделать запрос к классу `Win32_Processor` (листинг 12.6).

**Листинг 12.6. Получение информации о процессорах**

```
private void butProc_Click(object sender, EventArgs e)
{
    // Соединяемся с удаленной машиной
```

```
ConnectionOptions options = new ConnectionOptions();
options.Username = @"gamma\admin";
options.Password = "mypass";

ManagementScope scope =
    new ManagementScope("\\\\srv-sql\\root\\cimv2", options);
scope.Connect();

// Делаем запрос к удаленной машине
WqlObjectQuery query =
    new WqlObjectQuery("Select * from Win32_Processor");

ManagementObjectSearcher find =
    new ManagementObjectSearcher(query);
string nl = Environment.NewLine;
int i = 0;

foreach (ManagementObject mo in find.Get())
{

    textBox1.Text +=
        ("-------------- Processor #" + i + " --------------") + nl;
    textBox1.Text +=
        ("Processor address width in bits............."
            + mo["AddressWidth"]) + nl;
    textBox1.Text +=
        ("Caption....................................."
            + mo["Caption"]) + nl;
    textBox1.Text +=
        ("Processor address width in bits............."
            + mo["AddressWidth"]) + nl;
    textBox1.Text +=
        ("Current clock speed (in MHz)................."
            + mo["CurrentClockSpeed"]) + nl;
    textBox1.Text +=
        ("Processor data width........................"
            + mo["DataWidth"]) + nl;
    textBox1.Text +=
        ("Unique string identification................."
            + mo["DeviceID"]) + nl;
```

```
textBox1.Text +=
    ("External clock frequency....................."
        + mo["ExtClock"]) + nl;
textBox1.Text +=
    ("Processor data width........................."
        + mo["DataWidth"]) + nl;
textBox1.Text +=
    ("L2 cache size................................"
        + mo["L2CacheSize"]) + nl;
textBox1.Text +=
    ("L2 cache speed..............................."
        + mo["L2CacheSpeed"]) + nl;
textBox1.Text +=
    ("Load percentage (average value for second)..."
        + mo["LoadPercentage"]) + nl;
textBox1.Text +=
    ("Manufacturer................................."
        + mo["Manufacturer"]) + nl;
textBox1.Text +=
    ("Maximum speed (in MHz)......................."
        + mo["MaxClockSpeed"]) + nl;
textBox1.Text +=
    ("Name........................................."
        + mo["Name"]) + nl;
textBox1.Text +=
    ("Support for power management................."
        + mo["PowerManagementSupported"]) + nl;
textBox1.Text +=
    ("Unique identificator describing processor...."
        + mo["ProcessorId"]) + nl;
textBox1.Text +=
    ("Role (CPU/math).............................."
        + mo["Role"]) + nl;
textBox1.Text +=
    ("Socket designation..........................."
        + mo["SocketDesignation"]) + nl;
textBox1.Text +=
    ("Status......................................."
        + mo["Status"]) + nl;
textBox1.Text +=
    ("Processor version............................"
        + mo["Version"]) + nl;
```

```
        textBox1.Text +=
            ("Socket voltage..............................."
                + mo["VoltageCaps"]) + nl;
        i++;
    }
}
```

# Информация о свойствах видеоконтроллера

Пространство имен `Win32_VideoController` дает доступ к свойствам видеоконтроллера — параметры процессора, видеопамяти, разрешение экрана и частоту обновления (листинг 12.7).

**Листинг 12.7. Получение информации о свойствах видеоконтроллера**

```
private void butVideo_Click(object sender, EventArgs e)
{
    ConnectionOptions options = new ConnectionOptions();
    // ваш домен и учетная запись
    options.Username = @"domen\administrator";
    // ваш пароль
    options.Password = "yourpassword";

    ManagementScope scope =
       new ManagementScope("\\\\buh\\root\\cimv2", options);
    scope.Connect();

    // Запрашиваем информацию о видеоконтроллере
    ObjectQuery query = new ObjectQuery(
        "SELECT * FROM Win32_VideoController");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject m in queryCollection)
    {
        // Выводим информацию на экран
        string nl = Environment.NewLine;

        textBox1.Text = "Имя : " + m["Name"] + nl;
        textBox1.Text += "Процессор : " + m["VideoProcessor"] + nl;
        textBox1.Text += "Видеопамять: " + m["AdapterRam"] + nl;
```

```
        textBox1.Text += "Разрешение: " + m["VideoModeDescription"] + nl;
        textBox1.Text += "Частота обновления : "
            + m["CurrentRefreshRate"];
    }
}
```

# Получение свойств приводов компакт-дисков

Теперь мы хотим узнать немного о приводах компакт-дисков. Для этого существует класс `Win32_CDROMDrive`. Его применение показано в листинге 12.8.

**Листинг 12.8. Получение информации о свойствах приводов компакт-дисков**

```
private void butCDROM_Click(object sender, EventArgs e)
{
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"gamma\admin";
    options.Password = "mypass";

    ManagementScope scope =
       new ManagementScope("\\\\bron5\\root\\cimv2", options);
    scope.Connect();

    // Запрашиваем информацию о приводах компакт-дисков
    ObjectQuery query = new ObjectQuery(
        "SELECT * FROM Win32_CDROMDrive");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject mo in queryCollection)
    {
        string nl = Environment.NewLine;
        // Выводим информацию с удаленного компьютера
        textBox1.Text = "Описание: " + mo["Description"] + nl;
        textBox1.Text += "Диск: " + mo["Drive"] + nl;
        textBox1.Text += "Тип: " + mo["MediaType"] + nl;
        textBox1.Text += "Статус: " + mo["Status"] + nl;
    }
}
```

# Информация о параметрах загрузки Windows

Также существует возможность узнать параметры загрузки Windows при помощи класса `Win32_BootConfiguration` (листинг 12.9).

**Листинг 12.9. Получение параметров загрузки Windows**

```
private void button10_Click(object sender, EventArgs e)
{
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"domen\admin";
    options.Password = "12345";

    ManagementScope scope =
       new ManagementScope(
           "\\\\srv-sql\\root\\cimv2", options);
    scope.Connect();

    // Запрашиваем информацию о параметрах загрузки Windows
    ObjectQuery query = new ObjectQuery(
        "SELECT * FROM Win32_BootConfiguration");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject mo in queryCollection)
    {
        string nl = Environment.NewLine;
        // Выводим информацию с удаленного компьютера
        textBox1.Text = "Загрузочная папка : "
            + mo["BootDirectory"] + nl;
        textBox1.Text += "Описание: " + mo["Description"] + nl;
        textBox1.Text += "Последний диск: " + mo["LastDrive"] + nl;
        textBox1.Text += "ScratchDirectory: "
            + mo["ScratchDirectory"] + nl;
        textBox1.Text += "Временная папка: " + mo["TempDirectory"] + nl;
    }
}
```

# Информация о сетевом адаптере

Класс `Win32_NetworkAdapter` позволяет извлечь информацию о сетевом адаптере (листинг 12.10).

**Листинг 12.10. Получение свойств сетевого адаптера**

```
private void butAdapter_Click(object sender, EventArgs e)
{
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"gamma\admin";
    options.Password = "password";

    ManagementScope scope =
       new ManagementScope("\\\\metodist\\root\\cimv2", options);
    scope.Connect();

    // Запрашиваем информацию о сетевом адаптере
    ObjectQuery query = new ObjectQuery(
        "SELECT * FROM Win32_NetworkAdapter");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject mo in queryCollection)
    {
        string nl = Environment.NewLine;
        // Выводим информацию с удаленного компьютера
        textBox1.Text = "Производитель: " + mo["Manufacturer"] + nl;
        textBox1.Text += "MACAddress: " + mo["MACAddress"] + nl;
        textBox1.Text += "ProductName: " + mo["ProductName"] + nl;
        textBox1.Text += "AdapterType: " + mo["AdapterType"] + nl;
        textBox1.Text += "CreationClassName: "
            + mo["CreationClassName"] + nl;
    }
}
```

# Информация о мониторе

Класс `Win32_Monitors` возвращает информацию о мониторах (листинг 12.11).

**Листинг 12.11. Получение свойств монитора**

```
private void butMonitor_Click(object sender, EventArgs e)
{
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"gamma\admin";
    options.Password = "password";

    ManagementScope scope =
       new ManagementScope("\\\\smena01\\root\\cimv2", options);
    scope.Connect();

    // Запрашиваем информацию о мониторе
    ObjectQuery query = new ObjectQuery(
        "SELECT * FROM Win32_DesktopMonitor");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject mo in queryCollection)
    {
        string nl = Environment.NewLine;
        // Выводим информацию с удаленного компьютера
        textBox1.Text = "Описание: " + mo["Description"] + nl;
        textBox1.Text += "Тип монитора: " + mo["MonitorType"] + nl;
    }
}
```

# Материнская плата

Теперь чуть-чуть узнаем о материнской плате при помощи класса `Win32_BaseBoard` (листинг 12.12).

**Листинг 12.12. Получение свойств материнской платы**

```
private void butBoard_Click(object sender, EventArgs e)
{
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"gamma\admin";
    options.Password = "password";

    ManagementScope scope =
       new ManagementScope("\\\\smena01\\root\\cimv2", options);
    scope.Connect();

    // Запрашиваем информацию о материнской плате
    ObjectQuery query = new ObjectQuery(
        "SELECT * FROM Win32_BaseBoard");
    ManagementObjectSearcher searcher =
        new ManagementObjectSearcher(scope, query);

    ManagementObjectCollection queryCollection = searcher.Get();
    foreach (ManagementObject mo in queryCollection)
    {
        string nl = Environment.NewLine;
        // Выводим информацию с удаленного компьютера
        textBox1.Text = "Manufacturer : " + mo["Manufacturer"] + nl;
        textBox1.Text += "Name: " + mo["Name"] + nl;
    }
}
```

# Вывод списка общих ресурсов

Класс `Win32_Share` позволяет получить информацию об общих ресурсах системы, которые могут совместно использоваться всеми участниками сети (листинг 12.13). К подобным ресурсам относятся каталоги, устройства, принтеры, съемные носители и др.

**Листинг 12.13. Получение списка общих ресурсов**

```
private void butShare_Click(object sender, EventArgs e)
{
    // Соединяемся с удаленной машиной
    ConnectionOptions options = new ConnectionOptions();
```

```
    options.Username = @"gamma\admin";
    options.Password = "password";

    ManagementScope scope =
        new ManagementScope("\\\\mymachine\\root\\cimv2", options);
    scope.Connect();

    // Делаем запрос к удаленной машине
    System.Management.ObjectQuery oq =
        new System.Management.ObjectQuery("SELECT * FROM Win32_Share");

    ManagementObjectSearcher find =
         new ManagementObjectSearcher(scope, oq);

    string nl = Environment.NewLine;

    foreach (ManagementObject mo in find.Get())
    {
       textBox1.Text=("Список общих ресурсов = " + mo["Name"]) + nl;
    }
}
```

# Информация о логических дисках

С помощью класса `Win32_LogicalDisk` можно узнать информацию о локальных дисках на компьютере (листинг 12.14).

**Листинг 12.14. Получение информации о логических дисках**

```
private void butLogDisk_Click(object sender, EventArgs e)
{
    // Соединяемся с удаленной машиной
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"gamma\admin";
    options.Password = "pass";

    ManagementScope scope =
       new ManagementScope("\\\\support\\root\\cimv2", options);
    scope.Connect();
```

```
    // Делаем запрос к удаленной машине
    string cmiPath = @"\root\cimv2:Win32_LogicalDisk.DeviceID='C:'";

    ManagementObject mo = new ManagementObject(cmiPath);

    // Выводим информацию
    string nl = Environment.NewLine;
    textBox1.Text = "Описание: " + mo["Description"] + nl;

    textBox1.Text+="Файловая система: " + mo["FileSystem"] + nl;
    textBox1.Text+="Свободно: " + mo["FreeSpace"] + nl;
    textBox1.Text +="Размер диска: " + mo["Size"] + nl;
}
```

# Перезагрузка компьютера

С помощью WMI можно не только получать интересующую нас информацию, но и выполнять различные действия. Например, мы можем заставить удаленный компьютер перезагрузиться при помощи команды `Reboot` класса `Win32_OperatingSystem`, применение которой показано в листинге 12.15.

**Листинг 12.15. Перезагрузка удаленного компьютера**

```
private void butReboot_Click(object sender, EventArgs e)
{
    // Соединяемся с удаленным компьютером
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"domen\admin";
    options.Password = "password";

    ManagementScope scope =
       new ManagementScope("\\\\machine\\root\\cimv2", options);
    scope.Connect();

    // Делаем запрос к удаленной машине
    System.Management.ObjectQuery oq =
        new System.Management.ObjectQuery(
             "SELECT * FROM Win32_OperatingSystem");
    ManagementObjectSearcher query1 =
        new ManagementObjectSearcher(scope, oq);
    ManagementObjectCollection queryCollection1 = query1.Get();
```

```
    foreach (ManagementObject mo in queryCollection1)
    {
        string[] ss ={ "" };
        mo.InvokeMethod("Reboot", ss);
        this.Text = mo.ToString();
    }
}
```

***Примечание***

Пример работы с классами WMI находится в папке WMIDemo на прилагаемом диске.

# Дополнительный пример

В качестве бонуса я предлагаю вам изучить пример, написанный канадским программистом с иранскими корнями Алирезой Ширази (Alireza Shirazi).

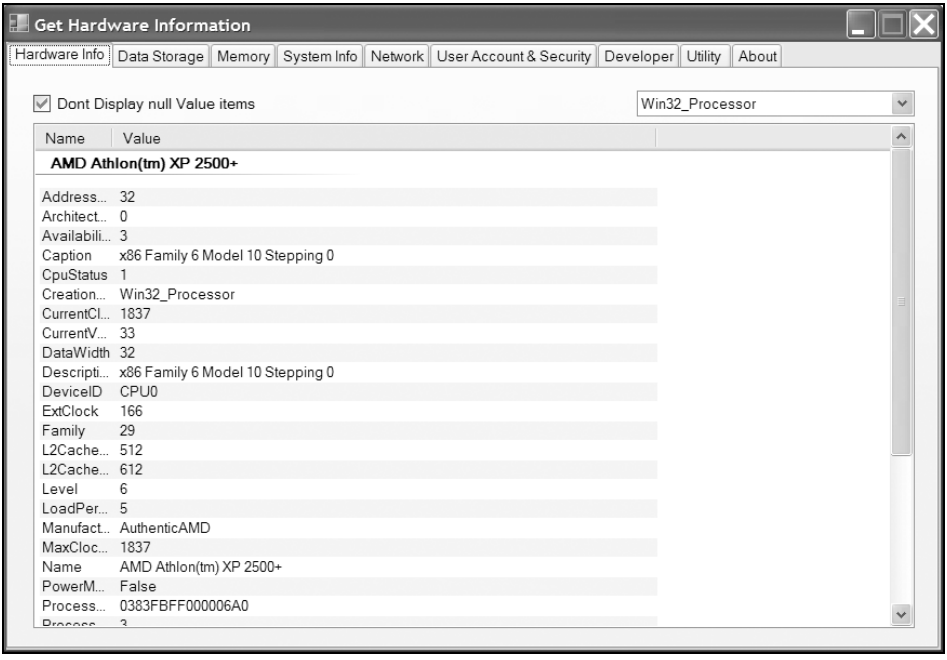


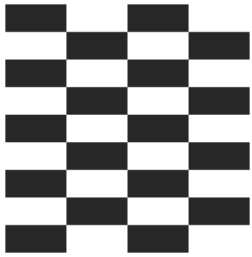

**Рис. 12.1.** Программа GetHardwareInfo

Автор выложил данный пример на сайте **http://www.codeproject.com**. Исходный код программы абсолютно бесплатен, и вы можете делать с ним, что хотите. Пример очень интересен, компактен и полезен. Я не буду приводить код программы в книге. Просто взгляните на скриншот этой программы (рис. 12.1), чтобы оценить ее возможности. Данный пример находится в папке GetHardwareInfo на прилагаемом диске. Кстати, на сайте CodeProject вы можете найти еще несколько примеров, связанных с технологией WMI. Воспользуйтесь поиском по сайту и изучите эти примеры.

## Заключение

Использование технологии WMI — ключ ко многим задачам, для решения которых требуется узнать какую-то информацию о системе, оборудовании, программном обеспечении. Всего насчитывается несколько сотен классов WMI, использование которых иногда значительно проще, чем написание неуправляемого кода. К тому же надо учитывать тот факт, что классы WMI можно использовать для работы с удаленными компьютерами.

# Глава 13



# Мультимедиа

## Звуковые сигналы

Звуковые сигналы — один из способов привлечь внимание пользователя к проблеме. Но надо помнить, что у пользователя могут быть отключены динамики. Однако есть еще системный динамик, который издает простейшие гудки, и который обычно подключен. Давайте рассмотрим все способы.

### Функция Windows API *Beep*

Функция Windows API `Beep` воспроизводит тоновый звук через внутренний динамик с различной частотой. Вы можете задавать частоту (в герцах) и продолжительность звука (в миллисекундах) для получения различных звуков, как показано в листинге 13.1.

**Листинг 13.1. Воспроизведение тонового звука через внутренний динамик**

```
[DllImport("kernel32.dll")]
public static extern bool Beep(int freq, int duration);

private void button1_Click(object sender, EventArgs e)
{
    Beep(500, 500);
}
```

### Функция Windows API *MessageBeep*

Существует еще одна функция Windows API `MessageBeep`, которая воспроизводит один из системных звуков (листинг 13.2). Системные звуки можно оп-

ределить при помощи апплета Звук в Панели управления. Функция также может воспроизводить звук встроенного динамика компьютера.

**Листниг 13.2. Использование функции `MessageBeep`**

```
[DllImport("user32.dll")]
public static extern int MessageBeep(uint n);

private void button2_Click(object sender, EventArgs e)
{
    MessageBeep(0x0);
}
```

## Функция *Beep* для Visual Basic

Разработчики на Visual Basic могут применять встроенную функцию `Beep`, которая также использует системный динамик, хотя ее возможности ограничены по сравнению с одноименной функцией Windows API `Beep`. Чтобы задействовать эту функцию в своем проекте, программисты C# должны установить ссылку на пространство имен `Microsoft.VisualBasic` (меню **Project | Add Reference…**). Пример применения показан в листинге 13.3.

**Листинг 13.3. Функция `Beep` из библиотеки Visual Basic**

```
private void button3_Click(object sender, EventArgs e)
{
    Microsoft.VisualBasic.Interaction.Beep();
}
```

> ***Примечание***
>
> Пример работы со звуками находится в папке BeepDemo на прилагаемом диске.

## Звуковые файлы

Рассмотренные выше примеры морально устарели. В .NET Framework 2.0 наконец-то появилась возможность проигрывать звуковые файлы WAV с помощью управляемого кода. Для этой цели было добавлено новое пространст-

во имен `System.Media`, в состав которого входит класс `SoundPlayer`. Его применение проиллюстрировано в листинге 13.4.

**Листинг 13.4. Воспроизведение звуковых файлов при помощи класса `SoundPlayer`**

```
private void button1_Click(object sender, EventArgs e)
{
    System.Media.SoundPlayer player = new System.Media.SoundPlayer();
    player.SoundLocation = @"c:\windows\media\tada.wav";
    player.Play();
}
```

# И снова о *Beep*

Кроме класса `SoundPlayer`, в пространство имен `System.Media` входит новый класс `SystemSounds`, который позволяет воспроизводить системные звуки. В листинге 13.5 приведет пример воспроизведения системного звука.

**Листинг 13.5. Воспроизведение системного звука `Beep`**

```
private void button2_Click(object sender, EventArgs e)
{
    System.Media.SystemSounds.Beep.Play();
}
```

Здесь читатель может удивленно воскликнуть, а зачем нам тогда рассказывали о функции `MessageBeep` или функции `Beep` пространства имен `Microsoft.VisualBasic`? Все очень просто. Я специально привожу различные способы для решения разных задач. Каждый программист находит свое решение для одной и той же задачи. Но, с другой стороны, чаще просматривайте исходные коды своих старых программ. Вряд ли есть смысл использовать неуправляемый код при вызове функции Windows API `MessageBeep`, если теперь есть возможность вызова управляемого метода `Beep` класса `SystemSounds`.

### *Примечание*

Пример работы со звуковыми файлами находится в папке SystemMedia на прилагаемом диске.

## Как проигрывать звуки разных форматов?

Мы рассмотрели способ воспроизведения звукового файла формата WAV при помощи класса `SoundPlayer`. Но существует масса других звуковых форматов, которые используются для прослушивания музыкальных треков. Среди распространенных музыкальных форматов лидерами являются форматы MP3 и WMA. Файлы этих форматов с помощью встроенных методов уже не проиграешь. Тем не менее, выход есть. Во-первых, можно воспользоваться помощью универсального проигрывателя Windows Media. Если подключить библиотеки данного проигрывателя, то можно воспроизводить файлы любого формата, который поддерживается этим проигрывателем. Чтобы подключить библиотеку Windows Media, выберите меню **Project | Add Reference…** и в открывшемся окне на вкладке **Browse** найдите файл wmp.dll, который находится в папке Windows\System32. После подключения этой библиотеки вам станут доступны многие свойства этого проигрывателя. Для примера в листинге 13.6 показан простой код для воспроизведения файла в формате WMA.

**Листинг 13.6. Воспроизведение звуков при помощи проигрывателя Windows Media**

```
private void button1_Click(object sender, EventArgs e)
{
    WMPLib.WindowsMediaPlayer WMP = new WMPLib.WindowsMediaPlayer();

    this.Text = WMP.versionInfo;
    WMP.URL =
     @"C:\Documents and Settings\All Users\Documents\My Music\music.wma";
    WMP.controls.play();
}
```

## Воспроизведение MIDI и MP3 через неуправляемый код

Нельзя забывать, что пользователь может отказаться от установки стандартного проигрывателя Windows Media и установить какой-нибудь альтернативный музыкальный проигрыватель. И нет никакой гарантии, что необходимая библиотека wmp.dll окажется на его компьютере. В этом случае нам опять поможет вызов неуправляемой функции Windows API `mciSendString` (листинг 13.7).

**Листинг 13.7. Воспроизведение MIDI через неуправляемый код**

```
[DllImport("winmm.dll", EntryPoint = "mciSendStringA",
CharSet = CharSet.Ansi)]
public static extern int mciSendString(string strCommand,
StringBuilder strReturnString, int cchReturn, IntPtr hwndCallback);

private void button2_Click(object sender, EventArgs e)
{
    string fileName = @"D:\MyBook\FunVB\15\PlaySound\town.MID";
    //  Открываем устройство с файлом town.MID
    //  player является идентификатором device_id.
    mciSendString("open " + fileName + " type sequencer alias player",
                  null, 0, IntPtr.Zero);
    mciSendString("play player", null, 0, IntPtr.Zero);

}

private void button3_Click(object sender, EventArgs e)
{
    //Закрываем файл TOWN.MID и устройство
    mciSendString("close player", null, 0, IntPtr.Zero);
}
```

***ПРИМЕЧАНИЕ***

Пример работы со звуками разных форматов находится в папке MusicPlayer на прилагаемом диске.

## Извлечение информации из файлов MP3

Вам наверняка известно, что файлы MP3 содержат в себе специальные теги, в которых находится различная информация об артисте, названии альбома, годе выпуска, жанре песни и т. д. Используя эти теги, пользователь может сортировать песни по своему вкусу, а также быстро находить нужную песню. Предлагаю вашему вниманию редактор MP3-файлов, который позволяет извлекать и записывать новую информацию в теги музыкальных файлов. Более подробную информацию о стандартах, применяемых в тегах MP3-файлов, поищите самостоятельно. В листинге 13.8 приводится класс, позволяющий извлекать и записывать теги в MP3-файл.

**Листинг 13.8. Работа с тегами MP3-файлов**

```
using System;
using System.Text;
using System.IO;

namespace MP3TagEditor
{
    struct MP3
    {
        public string filePath;
        public string fileName;
        public string fileComplete;

        public bool hasID3Tag;

        public string id3Title;
        public string id3Artist;
        public string id3Album;
        public string id3Year;
        public string id3Comment;
        public byte id3TrackNumber;
        public byte id3Genre;

        public MP3(string path, string name)
        {
            this.filePath = path;
            this.fileName = name;
            this.fileComplete = path + "\\" + name;

            this.hasID3Tag = false;
            this.id3Title = null;
            this.id3Artist = null;
            this.id3Album = null;
            this.id3Year = null;
            this.id3Comment = null;
            this.id3TrackNumber = 0;
            this.id3Genre = 0;
        }
    }
```

```
class MP3Tag
{

    public enum genres : byte
    {
        Blues,
        ClassicRock,
        Country,
        Dance,
        Disco,
        Funk,
        Grunge,
        HipHop,
        Jazz,
        Metal,
        NewAge,
        Oldies,
        Other,
        Pop,
        RnB,
        Rap,
        Reggae,
        Rock,
        Techno,
        Industrial,
        Alternative,
        Ska,
        DeathMetal,
        Pranks,
        Soundtrack,
        EuroTechno,
        Ambient,
        TripHop,
        Vocal,
        JazzFunk,
        Fusion,
        Trance,
        Classical,
        Instrumental,
        Acid,
        House,
        Game,
```

```
SoundClip,
Gospel,
Noise,
AlternRock,
Bass,
Soul,
Punk,
Space,
Mediative,
InstrumentalPop,
InstrumentalRock,
Ethnic,
Gothic,
Darkwave,
TechnoIndustrial,
Electronic,
PopFolk,
Eurodance,
Dream,
SouthernRock,
Comedy,
Cult,
Gangsta,
Top40,
ChristianRap,
PopFunk,
Jungle,
NativeAmerican,
Cabaret,
NewWave,
Psychadelic,
Rave,
Showtunes,
Trailer,
LoFi,
Tribal,
AcidPunk,
AcidJazz,
Polka,
Retro,
Musical,
RocknRoll,
```

```
    HardRock,
    None = 255
};

// читаем информацию из MP3-файла
public static void ReadMP3Tag(ref MP3 paramMP3)
{
    // Считываем последние 128 байт (теги ID3) в байтовый массив
    FileStream fs;
    fs = new FileStream(paramMP3.fileComplete, FileMode.Open);
    byte[] bBuffer = new byte[128];
    fs.Seek(-128, SeekOrigin.End);
    fs.Read(bBuffer, 0, 128);
    fs.Close();

    // Конвертируем массив байтов в строки
    //Encoding instEncoding = new ASCIIEncoding();
    Encoding encoding = Encoding.GetEncoding(1251);

    string id3Tag = encoding.GetString(bBuffer);

    // Если имеется строка TAG, тогда начинаем
    // извлекать информацию
    if (id3Tag.Substring(0, 3) == "TAG")
    {
        // название песни
        paramMP3.id3Title   = id3Tag.Substring(3, 30).Trim();
        // исполнитель
        paramMP3.id3Artist  = id3Tag.Substring(33, 30).Trim();
        // название альбома
        paramMP3.id3Album   = id3Tag.Substring(63, 30).Trim();
        // год выпуска
        paramMP3.id3Year    = id3Tag.Substring(93, 4).Trim();
        // комментарий
        paramMP3.id3Comment = id3Tag.Substring(97, 28).Trim();

        // Получим номер трека, если используется ID3 v1.1
        // Сравниваем предоследний байт на равенство нулю
        if (id3Tag[125] == 0)
            paramMP3.id3TrackNumber = bBuffer[126];
        else
            paramMP3.id3TrackNumber = 0;
```

```
            paramMP3.id3Genre = bBuffer[127];
            paramMP3.hasID3Tag = true;
        }
        else
        {
            // Если теги ID3 не найдены, то создаем пустые данные,
            // чтобы потом пользователь мог записать свои данные
            paramMP3.id3Title = "";
            paramMP3.id3Artist = "";
            paramMP3.id3Album = "";
            paramMP3.id3Year = "";
            paramMP3.id3Comment = "";
            paramMP3.id3TrackNumber = 0;
            paramMP3.id3Genre = 0;
            paramMP3.hasID3Tag = false;
        }
    }

    // Сохраняем теги в файле MP3
    public static void SaveMP3Tag(ref MP3 paramMP3)
    {
        // Удаляем любые пробелы
        paramMP3.id3Title = paramMP3.id3Title.Trim();
        paramMP3.id3Artist = paramMP3.id3Artist.Trim();
        paramMP3.id3Album = paramMP3.id3Album.Trim();
        paramMP3.id3Year = paramMP3.id3Year.Trim();
        paramMP3.id3Comment = paramMP3.id3Comment.Trim();

        // Не позволяем выходить строке за допустимые пределы длины
        if (paramMP3.id3Title.Length > 30)
            paramMP3.id3Title = paramMP3.id3Title.Substring(0, 30);
        if (paramMP3.id3Artist.Length > 30)
            paramMP3.id3Artist = paramMP3.id3Artist.Substring(0, 30);
        if (paramMP3.id3Album.Length > 30)
            paramMP3.id3Album = paramMP3.id3Album.Substring(0, 30);
        if (paramMP3.id3Year.Length > 4)
            paramMP3.id3Year = paramMP3.id3Year.Substring(0, 4);
        if (paramMP3.id3Comment.Length > 28)
            paramMP3.id3Comment =
                paramMP3.id3Comment.Substring(0, 28);
```

```
// Создаем новый тег ID3 (128 байт)
byte[] tagByteArray = new byte[128];
for (int i = 0; i < tagByteArray.Length; i++)
    // инициализация массива значением 0
    tagByteArray[i] = 0;

// Конвертируем массив байтов в строку кодировки Windows-1251
Encoding encoding = Encoding.GetEncoding(1251);

// Копируем "TAG" в массив
byte[] newByteArray = encoding.GetBytes("TAG");
Array.Copy(newByteArray, 0,
           tagByteArray, 0, newByteArray.Length);
// Копируем Title в массив
newByteArray = encoding.GetBytes(paramMP3.id3Title);
Array.Copy(newByteArray, 0,
           tagByteArray, 3, newByteArray.Length);
// Копируем Artist в массив
newByteArray = encoding.GetBytes(paramMP3.id3Artist);
Array.Copy(newByteArray, 0,
           tagByteArray, 33, newByteArray.Length);
// Копируем Album в массив
newByteArray = encoding.GetBytes(paramMP3.id3Album);
Array.Copy(newByteArray, 0,
           tagByteArray, 63, newByteArray.Length);
// Копируем Year в массив
newByteArray = encoding.GetBytes(paramMP3.id3Year);
Array.Copy(newByteArray, 0,
           tagByteArray, 93, newByteArray.Length);
// Копируем Comment в массив
newByteArray = encoding.GetBytes(paramMP3.id3Comment);
Array.Copy(newByteArray, 0,
           tagByteArray, 97, newByteArray.Length);

// Копируем Track и Genre в массив
tagByteArray[126] = paramMP3.id3TrackNumber;
tagByteArray[127] = paramMP3.id3Genre;

// Сохраняем результат на диске
FileStream fs =
```

```
                new FileStream(paramMP3.fileComplete, FileMode.Open);
            if (paramMP3.hasID3Tag)
                fs.Seek(-128, SeekOrigin.End);
            else
                fs.Seek(0, SeekOrigin.End);
            fs.Write(tagByteArray, 0, 128);
            fs.Close();
            paramMP3.hasID3Tag = true;
        }
    }
}
```

А в листинге 13.9 показан код, который позволяет просматривать и обновлять информацию, используя созданную вами в Visual Studio 2005/2008 форму.

**Листинг 13.9. Форма для работы с тегами MP3**

```
MP3 mp3file;

private void butLoadMP3_Click(object sender, EventArgs e)
{
    // Загружаем MP3-файл
    LoadMp3();
}

private void LoadMp3()
{
    // Выбираем файл MP3
    OpenFileDialog fileDialog = new OpenFileDialog();
    fileDialog.Filter = "файлы MP3 (*.mp3)|*.mp3";
    DialogResult result = fileDialog.ShowDialog();
    if (result == DialogResult.Cancel) return;
    string fileName = fileDialog.FileName;

    FileInfo fFileInfo = new FileInfo(fileName);
    mp3file = new MP3(fFileInfo.DirectoryName, fFileInfo.Name);
    MP3Tag.ReadMP3Tag(ref mp3file);

    // Выводим полученную информацию на экран
    txtTitle.Text = mp3file.id3Title;
    txtArtist.Text = mp3file.id3Artist;
```

```
    txtAlbum.Text = mp3file.id3Album;
    txtYear.Text = mp3file.id3Year;
    txtComment.Text = mp3file.id3Comment;
    txtGenre.Text = mp3file.id3Genre.ToString();
    cboGenres.SelectedIndex = mp3file.id3Genre;
}

private void SaveButton_Click(object sender, EventArgs e)
{
    SaveMP3();
}

private void SaveMP3()
{
    if (mp3file.id3Title == null) return;

    mp3file.id3Title = txtTitle.Text;
    mp3file.id3Artist = txtArtist.Text;
    mp3file.id3Album = txtAlbum.Text;
    mp3file.id3Year = txtYear.Text;
    mp3file.id3Comment = txtComment.Text;
    mp3file.id3Genre = Convert.ToByte(txtGenre.Text);

    MP3Tag.SaveMP3Tag(ref mp3file);
}

private void MP3Form_Load(object sender, EventArgs e)
{
    cboGenres.Items.Add("Блюз");
    cboGenres.Items.Add("Классический рок");
    cboGenres.Items.Add("Кантри");
    cboGenres.Items.Add("Dance");
    cboGenres.Items.Add("Диско");
    cboGenres.Items.Add("Фанк");
    cboGenres.Items.Add("Гранж");
    cboGenres.Items.Add("Хип-Хоп");
    cboGenres.Items.Add("Джаз");
    cboGenres.Items.Add("Метал");
    cboGenres.Items.Add("НьюЭйдж");
    cboGenres.Items.Add("Oldies");
    cboGenres.Items.Add("Другое");
    cboGenres.Items.Add("Поп");
```

```
    // дальше сами пишите
    cboGenres.Items.Add("Не определен");
}
```

Полный текст листинга вы найдете на диске. С помощью данного примера вы можете открыть любой файл MP3, считать информацию (если она есть) и при необходимости внести изменения в теги (рис. 13.1).

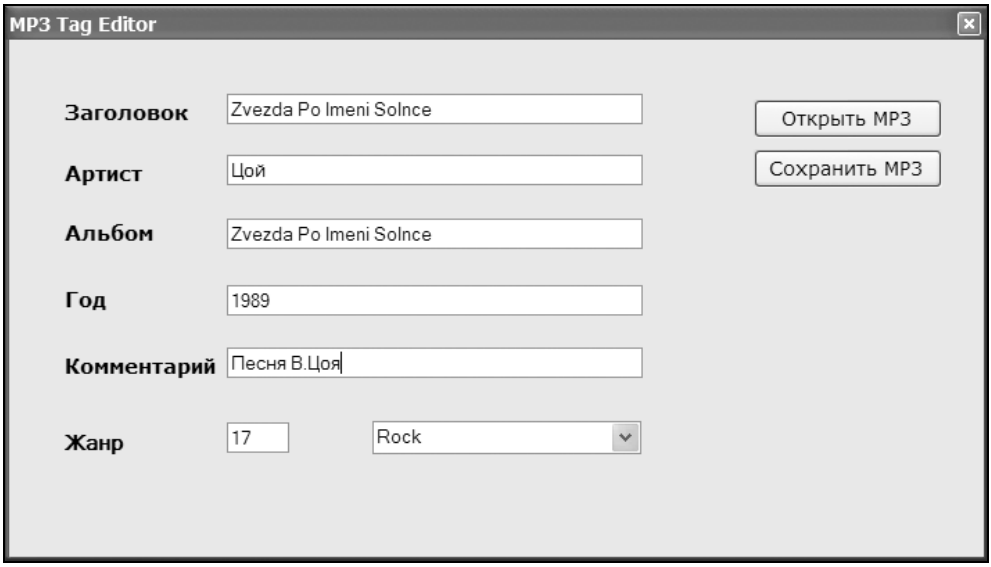



**Рис. 13.1.** Редактор тегов MP3-файлов

***Примечание***

Пример работы с тегами MP3 находится в папке MP3TagEditor на прилагаемом диске.

## Взаимодействие с Winamp

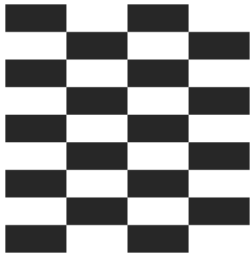
Среди поклонников музыки большой популярностью пользуется музыкальный проигрыватель Winamp. Сергей Борзов в своем блоге рассказывает, как сделать пользователя еще ленивее. В своей статье "WINAMP User…", кото-

рая находится по адресу **http://seregaborzov.wordpress.com/2007/10/17/winamp-user/**, автор рассказывает, как написать программу, позволяющую упростить управление проигрывателем. Я не стану приводить здесь его код. Во-первых, я не пользуюсь в данный момент проигрывателем Winamp, а во-вторых, я хочу, чтобы вы сами прочитали статью, которая содержит несколько интересных ссылок и полезных комментариев.

## Заключение

Использование мультимедийных возможностей в современном программировании — залог успеха многих программ у пользователей. Поэтому нужно серьезно подходить к этому вопросу при разработке различных обучающих, развлекательных и игровых приложений.

## Глава 14



# Разработка локализованных приложений

В этой главе будет затронута тема, которой, на мой взгляд, не уделяется должного внимания многими разработчиками программ. Мы поговорим о глобализации и локализации приложений для распространения их по всему миру. Для того чтобы понять, о чем идет речь, предлагаю взглянуть на следующий пример. Перед вами простое приложение, которое выводит на экран названия дней недели и месяцев на русском языке (рис. 14.1).

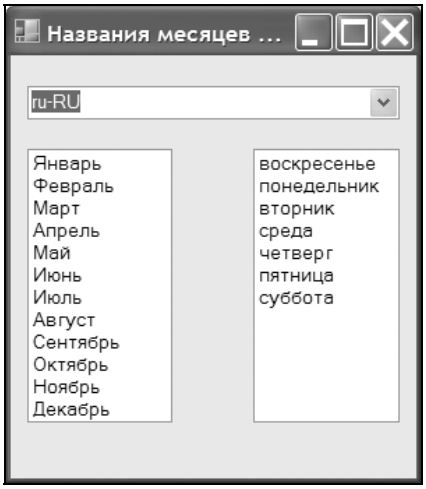



**Рис. 14.1.** Названия месяцев и недель на выбранном языке

Выберите теперь из списка другой язык, и вы увидите, что названия месяцев и дней недели изменились и теперь выводятся на другом языке. Вы можете

продолжить выбирать различные языки, и содержимое списков будет меняться. А теперь внимание. Я не знаю, как пишутся эти названия на немецком, английском, финском, украинском или французском языках. Более того, если посмотрите исходный код программы, который приведен в листинге 14.1, то не увидите этих названий в коде. То есть все эти названия уже прошиты в .NET Framework! С такой же легкостью я могу добавить в этот список эстонский, шведский, итальянский или арабский язык. Если ваша программа использует в работе дни недели и месяцев, то, используя эту возможность, вы можете в автоматическом режиме подстраиваться под родной язык пользователя на уровне операционной системы. Иными словами, вы можете воспользоваться возможностями глобализации, предоставляемыми .NET Framework.

**Листинг 14.1. Вывод локализованных названий месяцев и недель**

```
using System.Globalization;

private void FillListBox()
{
    string culture =  cboCulture.SelectedItem.ToString();
    CultureInfo ci = new CultureInfo(culture);
    DateTimeFormatInfo dtfi = new DateTimeFormatInfo();
    dtfi = ci.DateTimeFormat;

    DateTime dt;
    lstMonth.Items.Clear();

    for (int i = 1; i <= 12; i++)
    {
        dt = new DateTime(1900, i, 1);
        lstMonth.Items.Add(dt.ToString("MMMM", ci));
    }

    lstWeek.Items.Clear();

    for (DayOfWeek i = DayOfWeek.Sunday; i <= DayOfWeek.Saturday; i++)
    {
        lstWeek.Items.Add(dtfi.GetDayName(i));
    }
}
```

```
private void cboCulture_SelectedValueChanged(object sender, EventArgs e)
{
    FillListBox();
}

private void Form1_Load(object sender, EventArgs e)
{
    cboCulture.Items.Add("ru-RU");
    cboCulture.Items.Add("en-US");
    cboCulture.Items.Add("de-DE");
    cboCulture.Items.Add("fr-FR");
    cboCulture.Items.Add("uk-UA");
    cboCulture.Items.Add("fi-FI");
    cboCulture.SelectedIndex = 0;
}
```

***ПРИМЕЧАНИЕ***

Пример работы с названиями месяцев и недель находится в папке CultureInfo на прилагаемом диске.

# Общая информация о локализации

Вторая тема, которая тесно связана с глобализацией, — это локализация приложений. Становление современной компьютерной индустрии начиналось в США, поэтому неудивительно, что "родным" языком компьютеров стал английский язык. Операционные системы, интерфейс программ, конструкции языков программирования — все было на английском. Пока использование компьютеров было уделом специалистов, это не составляло проблемы. Но когда компьютерами и компьютерными технологиями (в первую очередь Интернетом) стали пользоваться широкие слои населения во всем мире, то возникла проблема локализации программ. Несмотря на все уверения опытных пользователей, что изучить несколько английских слов совсем не сложно, обычные пользователи (домохозяйки, школьники, бухгалтеры) не собирались следовать этим советам. Некоторые разработчики программ поняли, что локализация программ поможет им расширить рынок сбыта платных версий утилит. Сама компания Microsoft тоже стала выпускать локализованные версии своих продуктов, в том числе Windows и Office. Также появились и программисты-энтузиасты, которые самостоятельно переводили английский интерфейс программ на свой родной язык. Применительно к нашей стране это

направление получило название русификации программы. Русифицированные программы (официальные и программы-самоделки) стали пользоваться спросом у пользователей. Люди стали целенаправленно искать именно русифицированные версии любимых программ. Появились отдельные сайты, посвященные этой тематике. Требования пользователей к интерфейсу программ можно считать оправданными. Вряд ли бы вам понравилось иметь автомобиль, видеомагнитофон и холодильник, если на их панелях и в руководствах по эксплуатации использовался бы японский язык, только потому, что эта техника была собрана в Японии.

До поры до времени проблеме локализации не уделялось должного внимания. Каждый разработчик придумывал свои решения в этой области. С появлением .NET Framework разработка многоязыковых приложений стала проще и удобнее. Пока не все программисты знакомы с этой замечательной возможностью. Давайте поближе познакомимся с процессом создания локализованных приложений.

## Локализующие идентификаторы

До появления .NET Framework в Windows использовалось понятие локализующий идентификатор (locale identifier, LCID), который активно использовался в соответствующих функциях Windows API для получения информации об используемом пользователем региональном стандарте. Сам идентификатор LCID является 32-битным беззнаковым целым числом. Это число можно разбить на четыре части. Первые две части определяют язык и его подмножество, а последние две части задают порядок сортировки текстовых строк. В Windows используется LCID-идентификатор по умолчанию, который выбирается системой при установке. В дальнейшем данный идентификатор наследуется каждым пользователем системы и его действие распространяется на все потоки, создаваемые пользователем. Пользователь может в любой момент сменить LCID-идентификатор, используемый по умолчанию, через апплет **Язык и Региональные стандарты** в Панели управления, а разработчик может это сделать программным путем через вызов специальных функций Windows API.

С появлением .NET Framework LCID-идентификаторы были включены в библиотеку классов и являются частью пространства имен, связанных с культурой. Под термином культура в .NET Framework подразумевается язык и местонахождение пользователя.

# Культура

Рассмотрим понятие культур поподробнее. Идентификатор культуры (culture identifier) состоит и кода языка и кода страны/региона. Полный список идентификаторов культур можно посмотреть в документации .NET Framework SDK, а в табл. 14.1 мы приводим лишь небольшой список, взятый из этой документации.

***Таблица 14.1.*** *Список идентификаторов культур*

| Идентификатор культур | Имя культуры | Язык (код) | Страна/Регион (код) |
|---|---|---|---|
| 0x0009 | en | Английский (en) | |
| 0x0C09 | en-AU | Английский (en) | Австралия (AU) |
| 0x0409 | en-US | Английский (en) | США (US) |
| 0x000C | fr | Французский (fr) | |
| 0x040C | fr-FR | Французский (fr) | Франция (FR) |
| | fr-LU | Французский (fr) | Люксембург (LU) |
| 0x0019 | ru | Русский (ru) | |
| 0x0419 | ru-RU | Русский (ru) | Россия (RU) |

Обратите внимание, что имя культуры может состоять только из кода языка (маленькие буквы) или сочетания кода языка и кода страны/региона. Важно понимать разницу между этими культурами. Культуры, в которых указан только код языка, называются *нейтральными* (neutral) или *только языковыми* (language-only), а культуры, в которых используется код страны, называются *конкретными* (specific). При локализации программ важно учитывать тип культуры. Например, для вывода текстовых сообщений достаточно языка культуры. Но если ваше приложение будет оперировать символами валют, датами и т. п., то необходимо в этом случае использовать конкретную культуру. Как правило, культура для пользовательского интерфейса приложения задается системой неявно, на основе региональных стандартов, или явно, программным способом. Для каждого потока текущая культура определяется отдельно. Каждый поток наследует культуру, установленную Windows по умолчанию. Сменить ее можно, используя пространство имен `System.Threading`.

# Приложение Culture Explorer

Прежде чем мы изучим процесс создания локализованных приложений, давайте сначала рассмотрим пример, позволяющий глубже понять, как управлять культурами. В качестве основы я взял замечательный пример, опубликованный еще в 2002 году в журнале MSDN Magazine. И хотя этот пример был написан еще для Visual Studio .NET 2003, он без проблем конвертировался в новый проект под Visual Studio 2005. Так как прошло уже много времени, и не все имеют возможность прочитать эту статью, я постараюсь пересказать самые важные и интересные детали, в ней описанные. Кроме того, я внес в проект небольшие косметические изменения, но в целом программа осталась такой же, как было описано в статье.

С помощью этого приложения мы сможем просматривать установленные культуры, управлять текущей культурой в период выполнения, а также видеть в реальном времени, как влияет смена культуры на различные типы информации. Добавим на форму элементы управления `TreeView` и `ListView` и свяжем их разделителем (элемент управления `Splitter`). Для элемента `TreeView` установим свойства `Sorted` в `true` и `HideSelection` в `false`. Для элемента `ListView` свойство `View` установим в значение `Details` и свойство `GridLines` в `true`. Кроме того, добавим в `ListView` элементы `ColumnHeader`, в которых будет отображаться выводимая информация о культуре.

Теперь настало время заняться кодом программы. Сначала подключим к проекту два пространства имен:

```
using System.Globalization;
using System.Threading;
```

Теперь нам необходимо заполнить элемент `TreeView` информацией обо всех культурах. Сделаем это на этапе загрузки формы (листинг 14.2).

**Листинг 14.2. Заполнение элемента `TreeView` информацией о культурах**

```
private void Form1_Load(object sender, System.EventArgs e)
{
    TreeNode tempNode;

    // Добавляем к дереву нейтральные культуры, как корневые узлы
    foreach (CultureInfo CultureX in CultureIn-
fo.GetCultures(CultureTypes.NeutralCultures))
    {
        tempNode = new TreeNode(CultureX.EnglishName
                                + " [" + CultureX.Name + "]");
```

```
            tempNode.Tag = CultureX;
            treeView1.Nodes.Add(tempNode);
        }

        // Перебираем все конкретные культуры и добавляем
        // каждую из них к родительской нейтральной культуре в дереве
        foreach (CultureInfo CultureX in
                 CultureInfo.GetCultures(CultureTypes.SpecificCultures))
        {
            foreach (TreeNode NodeX in treeView1.Nodes)
            {
                if (NodeX.Tag.Equals(CultureX.Parent))
                {
                    tempNode = new TreeNode(CultureX.EnglishName
                                            + " [" + CultureX.Name + "]");
                    tempNode.Tag = CultureX;
                    tempNode.ForeColor = Color.Red;
                    foreach (CultureInfo CultureY in CultureInfo.GetCultures(
                                        CultureTypes.InstalledWin32Cultures))
                    {
                        if (CultureY.Equals(CultureX))
                        {
                            tempNode.ForeColor = Color.Black;
                            break;
                        }
                    }
                    NodeX.Nodes.Add(tempNode);
                    break;
                }
            }
        }
    }
```

В первом операторе `foreach` мы добавляем в дерево `TreeView` все нейтральные культуры в качестве корневых узлов. Экземпляр класса `CultureInfo` всегда представляет в .NET определенную культуру (нейтральную или конкретную). С помощью статического метода `GetCultures` класса `CultureInfo` мы можем получить список всех нейтральных культур, используя перечисление `CultureTypes`. На основе полученной информации формируем первый уровень дерева, последовательно добавляя каждую культуру. Далее мы создаем строку с описанием названия культуры, а саму культуру добавляем в свойст-

во Tag соответствующего узла. Следующий оператор foreach перебирает все конкретные культуры и добавляет каждую из них к родительской нейтральной культуре в дереве. Если точно определенная культура отсутствует в системе, то она отображается красным цветом.

Это была подготовительная работа. А основная работа приложения производится в обработчике события treeview1_AfterSelect, приведенном в листинге 14.3.

**Листинг 14.3. Заполнение элемента ListView информацией о выбранной культуре**

```
private void treeView1_AfterSelect(object sender,
    System.Windows.Forms.TreeViewEventArgs e)
{
    CultureInfo selectedCulture;
    selectedCulture = (CultureInfo)treeView1.SelectedNode.Tag;
    if (selectedCulture.IsNeutralCulture == true)
    {
        listView1.Items[3].SubItems[1].Text = "Нейтральная";
        for (int x = 4; x <= 10; x++)
            listView1.Items[x].SubItems[1].Text = "Нейтральная культура";
    }
    else
    {
        Thread.CurrentThread.CurrentCulture = selectedCulture;
        listView1.Items[3].SubItems[1].Text = "Конкретная";
        listView1.Items[4].SubItems[1].Text =
                  (DateTime.Now).ToShortDateString();
        listView1.Items[5].SubItems[1].Text =
                  (DateTime.Now).ToLongDateString();
        listView1.Items[6].SubItems[1].Text =
                  (DateTime.Now).ToShortTimeString();
        listView1.Items[7].SubItems[1].Text =
                  (DateTime.Now).ToLongTimeString();
        listView1.Items[8].SubItems[1].Text = (35500.75).ToString("n");
        listView1.Items[9].SubItems[1].Text = (1750.25).ToString("c");
        listView1.Items[10].SubItems[1].Text =
                  (selectedCulture.Calendar.ToString()).Remove(0,21);
    }
        listView1.Items[0].SubItems[1].Text = selectedCulture.Name;
```

```
        listView1.Items[1].SubItems[1].Text =
                    selectedCulture.EnglishName;
        listView1.Items[2].SubItems[1].Text = selectedCulture.NativeName;
}
```

В этом коде сначала мы получаем экземпляр класса `CultureInfo`, хранящийся в свойстве `Tag` узла:

```
selectedCulture = (CultureInfo)treeView1.SelectedNode.Tag;
```

Получив интересующие нас данные, мы можем теперь менять текущую культуру в приложении и наблюдать, как смена культуры влияет на данные, показываемые в элементе `ListView`. Необходимо помнить, что в потоке в качестве текущей культуры можно устанавливать только конкретную культуру, так как установка нейтральной культуры вызывает исключение. Поэтому делаем соответствующую проверку, а затем меняем текущую культуру в потоке, в котором работает приложение. После этого мы заполняем данными `ListView`. Во время выполнения программы вы увидите, что при смене культуры автоматически меняются названия месяцев, валюты и другие данные. Обратите внимание, что в приложении Culture Explorer некоторые культуры помечены красным цветом. Это не установленные в вашей системе культуры. Тем не менее, их тоже можно выбрать для просмотра, только при этом некоторые элементы могут выводиться некорректно, если соответствующие символы выбранного языка недоступны.

> ***Примечание***
>
> Пример работы с культурами находится в папке CultureExplorer на прилагаемом диске.

# Разработка многоязычного приложения

Теперь, когда мы достаточно узнали о культурах, можно приступать к разработке многоязыкового приложения. Visual Studio 2005 позволяет легко создавать приложения, в которых интерфейс будет легко адаптироваться к региональным настройкам пользователя. Сам процесс создания многоязычного приложения делится на два этапа. Сначала создается обычное приложение стандартным способом. Далее пользовательский интерфейс приложения переводится на другие языки. Рассмотрим на конкретном примере. Запустите новый проект и добавьте на форму обычную кнопку с надписью **Good Morning**. Предположим, что нам надо перевести надпись на этой кнопке на разные

языки, чтобы программой могли воспользоваться пользователи из разных стран. Для начала перейдите в окно Solution Explorer и щелкните переключатель **Show All Files** (если он еще не выбран). Раскройте узел Form1.cs. Там вы можете увидеть файл Form1.resx. Это исходный файл ресурсов для формы, который компилируется в двоичный код ресурсов для создаваемого приложения. Теперь установите свойство формы **Localizable** в значение `True`, а свойству формы **Language** присвойте значение `German`. Далее измените текст на кнопке. Предположим, мы присвоим тексту кнопки значение **Guten Morgen**. Обратите внимание, что после этих действий к форме был добавлен дополнительный файл ресурса Form1.de-DE.resx. Повторите еще раз эту операцию, выбрав в качестве языка **Language** значение `Russian` и присвоив кнопке текст **Доброе утро**. Visual Studio 2005 добавит в проект еще один файл Form1.ru-RU.resx. Таким образом, у нас имеются фактически три разных экземпляра формы. Экземпляр, редактируемый в дизайнере форм, зависит от значения свойства `Language`, установленного в этот момент. Чтобы убедиться в этом, сделаем такую операцию. Как вы понимаете, надпись, будучи переведенной на другой язык, может не поместиться на кнопке заданного размера. Сделаем так: выберем русскоязычный экземпляр формы, сделаем размер кнопки квадратным и увеличим ее в размерах, а в текст кнопки внесем изменения: **Доброе утро, страна!** И новый текст, и новые размеры кнопки сохранятся только в русской версии формы. Например, если вы переключитесь в немецкую версию формы, то размер кнопки будет восстановлен до исходного состояния.

Теперь можно протестировать программу. Запустите проект, и у вас на экране должна появиться форма с русским интерфейсом (при условии, что у вас стоит локализованная русская версия Windows). Закройте программу. Внесите изменения в код программы. Перед вызовом метода `InitializeComponent` в конструкторе формы добавьте следующую строку:

```
System.Threading.Thread.CurrentThread.CurrentUICulture =
                                new System.Globalization.CultureInfo("de-DE");
```

Снова запустите проект, и вы теперь увидите немецкую версию программы! В принципе в использовании этого кода нет необходимости, поскольку в реальном приложении интерфейс будет выбран автоматически, исходя из региональных настроек Windows.

### *Примечание*

Пример создания многоязычного приложения находится в папке Locale на прилагаемом диске.

## Разделяй и властвуй

Итак, мы создавали несколько разных экземпляров формы и переводили названия кнопки для каждой из этих форм. Но, как правило, переводом и созданием программ занимаются разные люди. И совсем необязательно, чтобы переводчики имели доступ к исходному коду программы. Для решения этой задачи вам нужно воспользоваться утилитой Windows Resource Localization Editor, которая поставляется вместе с .NET SDK. Исполняемый файл утилиты WinRes.exe находится в папке bin пакета SDK. Сама утилита очень похожа на дизайнер форм в Visual Studio 2008 и очень проста в использовании. Открываете в программе нужный файл ресурсов resx и начинаете редактировать все элементы интерфейса. Закончив работу, передаете модифицированный файл разработчику, который добавляет его в проект. Таким образом, переводчику даже нет необходимости иметь сам пакет Visual Studio.

## Сопутствующие сборки

Теперь посмотрим, как использовать эти ресурсы в приложениях. Откройте проект в Visual Studio 2005, разверните каталог bin в окне **Solution Explorer** (при необходимости выберите **Show All Files**) и вы увидите, что в каталоге сборки (Debug или Release) присутствуют дополнительные каталоги. Эти каталоги создаются для каждой культуры, которую вы добавляли к форме. Их имена всегда соответствуют идентификатору культуры, и в этих каталогах находится файл ИмяПроекта.resources.dll. Вам также следует придерживаться подобной структуры директорий и схемы именования при установке собственных приложений. Эти динамические библиотеки DLL называются сопутствующими сборками (satellite assembles). Исполняющая среда при работе приложения использует эти файлы для нахождения в них нужных ресурсов. Предположим, что была выбрана русская культура ru-RU. Исполняющая среда ищет любые ресурсы, предусмотренные для этой культуры, в Global Assembly Cache (GAC). Если там необходимых ресурсов нет, то поиск продолжается в каталоге выполняемой в данный момент сборки, в котором исполняющая среда пытается обнаружить каталог с именем ru-RU. Далее в GAC идет поиск сборки ресурсов, соответствующий родителю текущей культуры. Исполняющая среда должна найти и эквивалентный каталог ru. Если поиск завершится успешно, то исполняющая среда будет использовать ресурсы из сопутствующей сборки в каталоге ru.

Если же запустить приложение в системе, где для текущей культуры (например, китайской) не будет предоставлено никаких ресурсов, исполняющая

среда не найдет ничего подходящего даже на четвертом этапе поиска. Тогда исполняющая среда будет использовать культуру по умолчанию. Культура по умолчанию — единственная, которая действительно скомпилирована в главную сборку.

Одно из основных преимуществ такого подхода к упаковке и выборке ресурсов состоит в том, что вы получаете возможность добавлять (или заменять) ресурсы для культур даже после установки приложения. Новые сопутствующие сборки размещаются в соответствующих подкаталогах или устанавливаются в GAC на компьютере пользователя. Перекомпиляция кода при этом не требуется. Таким образом, .NET Framework значительно упрощает разработку многоязыковых и локализованных приложений. Visual Studio 2008 в сочетании с WinRes позволяет создавать приложения Windows Forms, интерфейс которых легко адаптируется под язык, выбранный в операционной системе.

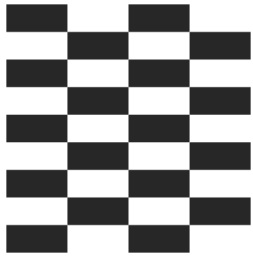
Кроме того, модель ресурсов, содержащихся в сопутствующих сборках, и поддержка Resource Fallback, встроенная в исполняющую среду, дают возможность разрабатывать приложения, которые готовы к применению в любой стране и которые можно превратить в действительно многоязыковые программы без модификации их кода или установки новых версий.

В завершение главы о локализации и глобализации хочу дать еще одну ссылку на эту тематику — на сайте RSDN выложена статья "Локализация" (**http://www.rsdn.ru/article/dotnet/csprof2005_localization.xml**), которая является отрывком из книги "Язык программирования C# 2005 для профессионалов".

# Заключение

Разработка локализованных приложений — насущная необходимость в современном мире, если разработчик хочет, чтобы программа была популярна у пользователей из многих стран. Visual Studio 2008 и .NET Framework предоставляют широкие и удобные возможности для создания таких программ.

## Глава 15



# Microsoft Office

Пакет Microsoft Office является популярным набором офисных программ, который широко используется во всем мире. Программисты могут использовать Microsoft Office в своих целях для решения многих задач. Рассмотрим несколько примеров использования программ Excel и Outlook и управления ими с помощью C#. При автоматизации таких приложений, как программы, входящие в состав Microsoft Office, вызовы их свойств и методов объектов должны быть определенным образом подключены к этим объектам. Процесс подключения вызовов свойств и методов к объектам, реализующим эти свойства и методы, обычно называется связыванием. В Visual C# существуют два типа связывания: раннее связывание (early binding) и позднее связывание (late binding). Выбранный программистом тип связывания повлияет на такие характеристики, как производительность, гибкость и удобство сопровождения. Мы рассмотрим оба варианта связывания. Вы можете также почитать на сайте Microsoft статью из Базы Знаний "Связывание с серверами автоматизации Office при помощи Visual C# .NET", которая находится по адресу **http://support.microsoft.com/kb/302902/ru**.

## Excel

### Раннее связывание

При раннем связывании осуществляется привязка непосредственно к методам и свойствам конкретного приложения Microsoft Office с использованием сведений о свойственных ему типах данных. При этом компилятор может проверить синтаксис и соответствие типов, правильность указания числа и типа параметров, передаваемых методу или свойству, и тип возвращаемого значения. Поскольку при выполнении приложения раннее связывание требует меньше времени для вызова свойств или методов, оно может работать бы-

стрее, чем позднее. Однако разница в производительности, как правило, незначительна.

При этом у раннего связывания есть один недостаток, который заключается в том, что оно может вызвать проблемы совместимости версий. Например, если в новой версии Microsoft Excel появился новый метод или свойство, которые были недоступны в предыдущей версии Microsoft Excel, и программист к нему обращается, то при работе с предыдущими версиями вероятно появление различных ошибок. Аналогичная ситуация возникнет, если были внесены изменения в существующие методы или свойства. Сама компания Microsoft рекомендует для предотвращения подобных проблем с ранним связыванием использовать сведения о типах для самой ранней версии поддерживаемого приложения Microsoft Office.

Рассмотрим пример создания проекта, использующего раннее связывание.

1. Создайте новый проект Windows Application.
2. Добавьте ссылку на библиотеку Microsoft Excel 12.0 Object Library (для этого вызовите меню **Project | Add Reference** и в открывшемся окне на вкладке **COM** выберите **Microsoft Excel 12.0 Object Library**).
3. Поместите на форму кнопку **Button1**.
4. Откройте редактор кода и добавьте код, приведенный в листинге 15.1.

**Листинг 15.1. Пример раннего связывания**

```
using System.Reflection;
using Excel = Microsoft.Office.Interop.Excel;

private void button1_Click(object sender, System.EventArgs e)
{
    Excel.Application objApp;
    Excel._Workbook objBook;
    Excel.Workbooks objBooks;
    Excel.Sheets objSheets;
    Excel._Worksheet objSheet;
    Excel.Range range;

    try
    {
        // Создание экземпляра Excel и запуск новой рабочей книги
        objApp = new Excel.Application();
        objBooks = objApp.Workbooks;
```

```
        objBook = objBooks.Add(Missing.Value);
        objSheets = objBook.Worksheets;
        objSheet = (Excel._Worksheet)objSheets.get_Item(1);
              // Присваиваем текст ячейке C3
        range = objSheet.get_Range("C3", Missing.Value);

        range.set_Value(Missing.Value, "C#.Народные советы");

        //Возвращение контроля над Excel пользователю.
        objApp.Visible = true;
        objApp.UserControl = true;
    }
    catch(Exception theException)
    {
        String errorMessage;
        errorMessage = "Error: ";
        errorMessage = String.Concat(errorMessage, theException.Message);
        errorMessage = String.Concat(errorMessage, " Line: ");
        errorMessage = String.Concat(errorMessage, theException.Source);

        MessageBox.Show(errorMessage, "Error");
    }
}
```

В этом примере мы устанавливали ссылку на библиотеку Microsoft Excel 12.0 Object Library, которая относится к Microsoft Excel 2007. Если у вас установлена другая версия MS Excel, то используйте соответствующий порядковый номер библиотеки.

***Примечание***

Пример находится в папке EarlyBinding на прилагаемом диске.

## Автоматизация Excel или как работать массивами

В статье из Базы Знаний по адресу **http://support.microsoft.com/kb/302096/** находится статья, в которой рассказывается о том, как использовать автоматизацию Excel для работы с массивами. Чтобы не заполнять ряд ячеек по одному, вы можете установить свойство `Value` объекта `Range` равным двумер-

ному массиву. Аналогично, вы можете получить двумерный массив значений для ячеек за один раз при помощи свойства `Value`. Покажем это на конкретном примере.

1. Создайте новый проект и добавьте ссылку на библиотеку Microsoft Excel 12.0 Object Library (вызовите команду меню **Project** | **Add Reference** и на вкладке **COM** в открывшемся окне выделите в списке строку **Microsoft Excel 12.0 Object Library**).
2. Добавьте на форму две кнопки.
3. Добавьте на форму элемент управления `CheckBox` с именем и текстом `FillWithStrings`.
4. Напишите код, приведенный в листинге 15.2.

**Листинг 15.2. Пример работы с массивами в Excel**

```
using System.Reflection;
using Excel = Microsoft.Office.Interop.Excel;

Excel.Application objApp;
Excel._Workbook objBook;

private void button1_Click(object sender, EventArgs e)
{
    Excel.Workbooks objBooks;
    Excel.Sheets objSheets;
    Excel._Worksheet objSheet;
    Excel.Range range;

    try
    {
        // Создание экземпляра Excel и запуск новой рабочей книги
        objApp = new Excel.Application();
        objBooks = objApp.Workbooks;
        objBook = objBooks.Add(Missing.Value);
        objSheets = objBook.Worksheets;
        objSheet = (Excel._Worksheet)objSheets.get_Item(1);

        // Получение области (range), начальная ячейка которой имеет
        // адрес m_sStartingCell размером m_iNumRows x m_iNumCols
        range = objSheet.get_Range("A1", Missing.Value);
        range = range.get_Resize(5, 5);
```

```
if (this.FillWithStrings.Checked == false)
{
    // Создаем массив
    double[,] saRet = new double[5, 5];

    // Заполняем массив
    for (long iRow = 0; iRow < 5; iRow++)
    {
        for (long iCol = 0; iCol < 5; iCol++)
        {
            // Помещаем в ячейки счетчики
            saRet[iRow, iCol] = iRow * iCol;
        }
    }

    // Устанавливаем значение области равным массиву
    range.set_Value(Missing.Value, saRet);
}

else
{
    // Создаем массив
    string[,] saRet = new string[5, 5];

    // Заполняем массив
    for (long iRow = 0; iRow < 5; iRow++)
    {
        for (long iCol = 0; iCol < 5; iCol++)
        {
            // Помещаем адрес в ячейку
            saRet[iRow, iCol] =
                iRow.ToString() + "|" + iCol.ToString();
        }
    }

    // Устанавливаем значение области равным массиву
    range.set_Value(Missing.Value, saRet);
}

// Возвращаем контроль над Excel пользователю
objApp.Visible = true;
objApp.UserControl = true;
```

```
    }
    catch (Exception theException)
    {
        String errorMessage;
        errorMessage = "Ошибка: ";
        errorMessage = String.Concat(errorMessage, theException.Message);
        errorMessage = String.Concat(errorMessage, " Строка: ");
        errorMessage = String.Concat(errorMessage, theException.Source);

        MessageBox.Show(errorMessage, "Ошибка");
    }
}

private void button2_Click(object sender, EventArgs e)
{
    Excel.Sheets objSheets;
    Excel._Worksheet objSheet;
    Excel.Range range;

    try
    {
        try
        {
            // Получаем ссылку на первый лист в книге
            objSheets = objBook.Worksheets;
            objSheet = (Excel._Worksheet)objSheets.get_Item(1);
        }

        catch (Exception theException)
        {
            String errorMessage;
            errorMessage = "Не могу найти книгу Excel. Попробуйте нажать"
                + " Кнопку1, чтобы создать книгу Excel с данными"
                + " перед тем, как нажимать на Кнопку2.";

            MessageBox.Show(errorMessage, "Нет рабочей книги?");

            // Невозможно автоматизировать Excel, если он не находит
            // книги с данными, так что покинем процедуру
            return;
        }
```

```
        // Получаем область данных
        range = objSheet.get_Range("A1", "E5");

        // Получаем данные из области
        Object[,] saRet;
        saRet = (System.Object[,])range.get_Value(Missing.Value);

        // Определяем размеры массива
        long iRows;
        long iCols;
        iRows = saRet.GetUpperBound(0);
        iCols = saRet.GetUpperBound(1);

        // Создаем строку, содержащую все данные из области
        String valueString;
        valueString = "Array Data\n";

        for (long rowCounter = 1; rowCounter <= iRows; rowCounter++)
        {
            for (long colCounter = 1; colCounter <= iCols; colCounter++)
            {

                // Добавляем следующее значение к строке
                valueString = String.Concat(valueString,
                    saRet[rowCounter, colCounter].ToString() + ", ");
            }

            // Записываем символ перевода каретки
            valueString = String.Concat(valueString, "\n");
        }

        // Показываем значение массива
        MessageBox.Show(valueString, "Array Values");
    }

    catch (Exception theException)
    {
        String errorMessage;
        errorMessage = "Ошибка: ";
        errorMessage = String.Concat(errorMessage, theException.Message);
        errorMessage = String.Concat(errorMessage, " Строка: ");
        errorMessage = String.Concat(errorMessage, theException.Source);
```

```
            MessageBox.Show(errorMessage, "Ошибка");
        }
    }
```

Проверим наш пример в действии. Запустите проект. Щелкните на первой кнопке. Приложение запустит Microsoft Excel, откроет в нем новую книгу и заполнит ячейки первого листа в диапазоне A1:E5 числовыми данными из массива. Далее щелкните на второй кнопке. Приложение получит данные из ячеек A1:E5, занесет их в массив и покажет результат в окне сообщения. Теперь поставьте галочку в `FillWithStrings` и снова нажмите на первую кнопку. Теперь ячейки будут заполнены строковыми данными.

> ***ПРИМЕЧАНИЕ***
>
> Пример работы с массивами находится в папке ExcelDemo на прилагаемом диске.

## Позднее связывание

В отличие от раннего связывания, позднее связывание сопоставляет вызовы свойств и методов соответствующим объектам в процессе выполнения программы. Для этого в требуемых объектах должен быть реализован специальный интерфейс COM `IDispatch`. Метод `IDispatch::GetIDsOfNames` позволяет запрашивать объект о поддерживаемых методах и свойствах, а метод `IDispatch::Invoke` позволяет вызывать эти свойства и методы. Это дает возможность при использовании позднего связывания избежать некоторых зависимостей от версии, присущих раннему связыванию. Недостатком позднего связывания является то, что оно не позволяет проверять целостность кода автоматизации на этапе компиляции и не поддерживает возможности `Intellisense`, которые могут предоставить сведения для правильного вызова методов и свойств. Для использования позднего связывания Visual C# необходимо применить метод `System.Type.InvokeMember`. Этот метод вызывает методы `IDispatch::GetIDsOfNames` и `IDispatch::Invoke` для привязки к методам и свойствам сервера автоматизации.

Рассмотрим пример создания клиента автоматизации, использующего позднее связывание (листинг 15.3).

1. Запустите новый проект и добавьте на форму кнопку **Button1**.
2. Откройте окно редактора кода и добавьте код, приведенный в листинге 15.3.

**Листинг 15.3. Пример позднего связывания**

```
using System.Reflection;

private void button1_Click(object sender, System.EventArgs e)
{
    object objApp_Late;
    object objBook_Late;
    object objBooks_Late;
    object objSheets_Late;
    object objSheet_Late;
    object objRange_Late;
    object[] Parameters;

    try
    {
        // Получение типа класса и создание экземпляра Excel
        Type objClassType;
        objClassType = Type.GetTypeFromProgID("Excel.Application");
        objApp_Late = Activator.CreateInstance(objClassType);

        //Получение коллекции рабочих книг
        objBooks_Late = objApp_Late.GetType().InvokeMember "Workbooks",
            BindingFlags.GetProperty, null, objApp_Late, null);

        //Добавление новой рабочей книги
        objBook_Late = objBooks_Late.GetType().InvokeMember("Add",
            BindingFlags.InvokeMethod, null, objBooks_Late, null);

        //Получение коллекции рабочих листов
        objSheets_Late = objBook_Late.GetType().InvokeMember(
            "Worksheets", BindingFlags.GetProperty, null,
            objBook_Late, null);

        //Получение первого рабочего листа
        Parameters = new Object[1];
        Parameters[0] = 1;
        objSheet_Late = objSheets_Late.GetType().InvokeMember("Item",
            BindingFlags.GetProperty, null, objSheets_Late, Parameters);

        //Получение объекта диапазона, содержащего ячейку A1
        Parameters = new Object[2];
```

```
        Parameters[0] = "A1";
        Parameters[1] = Missing.Value;
        objRange_Late = objSheet_Late.GetType().InvokeMember("Range",
            BindingFlags.GetProperty, null, objSheet_Late, Parameters);

        //Написать "Hello, World!" в ячейке A1
        Parameters = new Object[1];
        Parameters[0] = "Hello, World!";
        objRange_Late.GetType().InvokeMember("Value",
            BindingFlags.SetProperty, null, objRange_Late, Parameters );

        //Возвращение контроля над Excel пользователю
        Parameters = new Object[1];
        Parameters[0] = true;
        objApp_Late.GetType().InvokeMember("Visible",
            BindingFlags.SetProperty, null, objApp_Late, Parameters);
        objApp_Late.GetType().InvokeMember("UserControl",
            BindingFlags.SetProperty, null, objApp_Late, Parameters);
    }
    catch( Exception theException )
    {
        String errorMessage;
        errorMessage = "Error: ";
        errorMessage = String.Concat(errorMessage, theException.Message);
        errorMessage = String.Concat(errorMessage, " Line: ");
        errorMessage = String.Concat(errorMessage, theException.Source);

        MessageBox.Show(errorMessage, "Error");
    }
}
```

Как видите, при работе с поздним связыванием необходимо подключить пространство имен `System.Reflection`. Далее для запуска Excel мы пользуемся классом `Activator`, который входит в пространство имен `System`. После того как вы запустили Excel, вам становится доступна вся объектная модель Excel, в частности:

- `Workbooks` (коллекция рабочих книг);
- `Worksheets` (коллекция листов).

Имея в своем распоряжении ссылку на коллекцию рабочих книг, мы можем получить доступ к любой книге. У каждой книги, в свою очередь, есть кол-

лекция содержащихся в ней листов, ссылку на которую мы также должны получить для доступа к конкретному листу. Обратите внимание, что доступ к книгам и к листам мы можем получить как по их имени, так и по их порядковому номеру, причем нумерация книг и страниц в коллекции начинается с единицы, а не с нуля, как это обычно принято в .NET Framework. Для доступа к коллекциям книг и листов используется метод `InvokeMember`. Данный метод имеет несколько перегруженных версий. Интерес представляет второй параметр, который содержит флаг, характеризующий связывание. Как правило, используются следующие флаги:

- `BindingFlags.InvokeMethod` — найти метод, определить его точку входа и выполнить его, передав ему массив фактических параметров;
- `BindingFlags.GetProperty` — установить свойство;
- `BindingFlags.SetProperty` — получить значение свойства.

***Примечание***

Пример находится в папке LateBinding на прилагаемом диске.

# Outlook

## Как получить сообщения из папки Входящие?

Опять воспользуемся помощью статьи Базы Знаний "How to use the Microsoft Outlook Object Library to retrieve a message from the Inbox by using Visual C#" (**http://support.microsoft.com/kb/310258**) и напишем пример. Вы можете получить сообщения из папки Входящие при помощи библиотеки Microsoft Outlook 2007 Object Library. Запустите новый проект и добавьте ссылку на **Microsoft Outlook 12 Object Library** так же, как мы делали это с примерами для Excel. Далее в редакторе кода напишите код, приведенный в листинге 15.4.

**Листинг 15.4. Получение сообщений из папки Входящие**

```
using System;
using System.Reflection;

using Outlook = Microsoft.Office.Interop.Outlook;
```

```
private void button1_Click(object sender, EventArgs e)
{
    // Запускаем приложение Outlook и инициализируем его
    Outlook.Application oApp = new Outlook.Application();

    // Получаем пространство имен MAPI
    Outlook.NameSpace oNS = oApp.GetNamespace("mapi");

    // Заходим, используя профиль по умолчанию (без диалогового окна)
    oNS.Logon(Missing.Value, Missing.Value, false, true);

    // Альтернативный метод входа под конкретным именем
    // TODO: Если используется этот метод входа, надо указать правильное
    // имя профайла и закомментировать предыдущую строку
    //oNS.Logon("profilename",Missing.Value,false,true);

    // Получение папки Входящие (Inbox)
    Outlook.MAPIFolder oInbox =
        oNS.GetDefaultFolder(Outlook.OlDefaultFolders.olFolderInbox);

    // Получение коллекции Items в папке Inbox
    Outlook.Items oItems = oInbox.Items;

    // Получим первое сообщение
    // Поскольку коллекция Items folder может содержать объекты разных
    // типов, используем явное преобразование типов при присваивании
    Outlook.MailItem oMsg = (Outlook.MailItem)oItems.GetFirst();

    // Выводим некоторые общие свойства
    textBox1.Text = (oMsg.Subject);
    textBox1.Text += textBox1.Text + oMsg.SenderName;
    textBox1.Text += textBox1.Text + oMsg.ReceivedTime;
    textBox1.Text += textBox1.Text + oMsg.Body;

    // Проверим наличие приложений
    int AttachCnt = oMsg.Attachments.Count;
    textBox1.Text += textBox1.Text
        + ("Attachments: " + AttachCnt.ToString());

    // TODO: Если используется библиотека Microsoft Outlook 10.0 Object
    // Library, раскомментировать следующие строки
    /*if (AttachCnt > 0)
```

```
    {
        for (int i = 1; i <= AttachCnt; i++)
             Console.WriteLine(i.ToString() + "-"
                 + oMsg.Attachments.Item(i).DisplayName);
    }*/

    // TODO: Если используется библиотека Microsoft Outlook 11.0 Object
    // Library, раскомментировать следующие строки
    /*if (AttachCnt > 0)
    {
        for (int i = 1; i <= AttachCnt; i++)
        Console.WriteLine(i.ToString() + "-"
            + oMsg.Attachments[i].DisplayName);
    }*/


    // Показать сообщение
    oMsg.Display(true);  //в модальном окне

    //Выйти
    oNS.Logoff();

    //Явно освободить объекты
    oMsg = null;
    oItems = null;
    oInbox = null;
    oNS = null;
    oApp = null;
}
```

***ПРИМЕЧАНИЕ***

Пример, в котором получаются сообщения Outlook, находится в папке OutlookDemo на прилагаемом диске.

## Получение уведомлений о новых письмах

Другой пример позволит в вашем приложении отслеживать появление новых писем в Outlook (листинг 15.5). В основу примера легла статья из Базы Знаний "How to use the NewMail event or the NewMailEx event to monitor Outlook and to notify you in Visual C# .NET that new e-mail messages have arrived", на-

ходящаяся по адресу **http://support.microsoft.com/kb/895940**. Как и в предыдущем примере, добавляем ссылку на библиотеку Microsoft Outlook 12.0. Пример основан на обработке события `NewMailEx`, с помощью которого осуществляется мониторинг поступления новых писем в папку Входящие программы Microsoft Office Outlook 2007.

**Листинг 15.5. Получение уведомления о новых письмах**

```
using Microsoft.Office.Interop.Outlook;
using OutLookApp = Microsoft.Office.Interop.Outlook.Application;

private static void outLookApp_NewMailEx(string EntryIDCollection)
{
    MessageBox.Show("Вы получили новое письмо, чей EntryIDCollection: \n"
        + EntryIDCollection, "Новое письмо", MessageBoxButtons.OK);
}

private void Form1_Load(object sender, EventArgs e)
{
    // Создадим объект Outlook
    ApplicationClass outLookApp = new ApplicationClass();

    // Обработчик события для поступления новых писем
    outLookApp.NewMailEx += new
        ApplicationEvents_11_NewMailExEventHandler(outLookApp_NewMailEx);
    this.Text = "Ждем новых писем...";
}
```

***Примечание***

Пример с получением уведомлений находится в папке OutlookNewMailEx на прилагаемом диске.

# VSTO

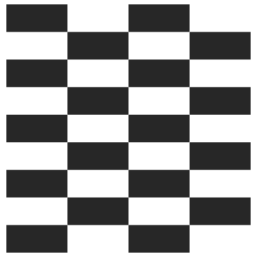
В Базе Знаний Microsoft я нашел еще несколько десятков статей, рассказывающих о взаимодействии пакета Microsoft Office с Visual Studio. Но эти примеры уходят в прошлое. Встречайте очередную новую революционную технологию — Visual Studio Tools for Office (VSTO). Этот набор инструментов доступен для загрузки с сайта Microsoft, чтобы использовать в Visual Stu-

dio 2005. А в Visual Studio 2008 данное дополнение уже является встроенным компонентом. Создание решений для Office с использованием этой технологии очень похоже на обычное программирование с Windows Forms. Но это уже тема для отдельного разговора.

## Заключение

Учитывая популярность пакета Microsoft Office, программисту нужно уметь писать программы, взаимодействующие с Word, Excel, Outlook и другими офисными приложениями. Думаю, что с распространением VSTO программирование для Office станет немного другим. Новая модель программирования выглядит многообещающе и привлекательно. Будущее покажет.

# Глава 16



# Локальная сеть и Интернет

Интернет-технологии по-прежнему являются очень динамично развивающимся направлением в компьютерной индустрии. Появляются новые протоколы, развиваются браузеры, появляются новые способы взаимодействия пользователей. Также развиваются местные локальные сети внутри предприятий. Поэтому сетевое программирование является очень важной частью в работе программиста. Знание некоторых приемов программирования сетей всегда пригодится любому разработчику.

В этой книге не будут рассматриваться аспекты работы с ASP.NET, так как изучение этой технологии достойно отдельной книги. Но мы рассмотрим более простые приемы.

## Информация о сети

Сначала рассмотрим некоторые вопросы, связанные с определением конфигурации сети, адресов компьютера и т. п.

### Как получить хост, порт, протокол из веб-адреса?

С помощью класса `System.UriBuilder` можно легко получить необходимую информацию о хосте, порте, используемом протоколе и другие данные. Пример показан в листинге 16.1.

**Листинг 16.1. Извлечение информации из веб-адреса**

```
private void button1_Click(object sender, EventArgs e)
{
    UriBuilder ubuild = new UriBuilder(@"http:\\rusproject.narod.ru:80");
```

```
    MessageBox.Show(ubuild.Host);
    MessageBox.Show(ubuild.Port.ToString());
    MessageBox.Show(ubuild.Scheme);
    MessageBox.Show(ubuild.Uri.ToString());
}
```

С помощью данного класса также можно создавать URI (универсальный идентификатор ресурсов) из составляющих элементов. Например, имеется адрес сайта, имя файла и фрагмент. Свойство `Fragment` содержит любой текст, который следует за маркером фрагмента (знаком `#`) в URI, включая сам маркер. При установлении свойства `Fragment` данный маркер также добавляется к его значению. В листинге 16.2 показано создание URI **http://rusproject.narod.ru/index.htm#main**.

**Листинг 16.2. Создание уникального идентификатора ресурсов**

```
UriBuilder builder = new UriBuilder("http://rusproject.narod.ru/");
builder.Path = "index.htm";
builder.Fragment = "main";

Uri myUri = builder.Uri;

MessageBox.Show(builder.ToString());
```

***Примечание***

Пример работы с классом `UriBuilder` находится в папке URI на прилагаемом диске.

## Как получить IP-адрес компьютера, используя DNS?

С помощью метода `GetHostEntry` класса `Net.Dns` можно получить IP-адрес компьютера, используя его имя, которые мы обычно вводим в браузер. Пример показан в листинге 16.3.

**Листинг 16.3. Получение DNS-имени компьютера**

```
private void butGetDNS_Click(object sender, EventArgs e)
{
    System.Net.IPHostEntry host;
```

```
    host = System.Net.Dns.GetHostEntry("yandex.ru");

    foreach (System.Net.IPAddress ip in host.AddressList)
    {
        MessageBox.Show(ip.ToString());
    }
}
```

В результате выполнения кода мы получим список IP-адресов указанного сайта. Теперь попробуйте ввести эти адреса непосредственно в браузер, чтобы убедиться в работоспособности кода. Также с помощью этого метода можно выполнять и обратное преобразование. Обратите внимание, что этот метод введен в .NET Framework 2.0, где он дополняет и заменяет устаревший метод `GetHostByName` из библиотеки классов .NET Framework 1.1.

## Как получить NETBIOS-имя машины?

Для получения имени NETBIOS локальной машины используется свойство `Envinronment.MachineName` (листинг 16.4).

**Листинг 16.4. Получение NETBIOS-имени компьютера**

```
private void butGetNETBIOS_Click(object sender, EventArgs e)
{
        MessageBox.Show(Environment.MachineName);
}
```

## Как получить IP-адрес локальной машины?

Для получения IP-адреса локальной машины достаточно указать пустую строку при использовании метода `GetHostEntry` (листинг 16.5).

**Листинг 16.5. Получение IP-адреса локальной машины**

```
 private void butGetLocalIP_Click(object sender, EventArgs e)
{
    string strIP = "";
    System.Net.IPHostEntry host;
    host = System.Net.Dns.GetHostEntry(strIP);
```

```
    foreach (System.Net.IPAddress ip in host.AddressList)
    {
        MessageBox.Show(ip.ToString());
    }
}
```

***ПРИМЕЧАНИЕ***

Примеры работы с сетевыми адресами находятся в папке DNS на прилагаемом диске.

## Ping

В состав операционной системы Windows входит утилита командной строки ping.exe, позволяющая определить доступность компьютера в сети. Для реализации подобной функциональности в .NET Framework 1.1 требовалось писать сложный код с использованием вызовов функций Windows API. В .NET Framework 2.0 появился новый класс `Ping` из пространства имен `System.Net.NetworkInformation`, с помощью которого реализация возможностей, предоставляемых утилитой ping.exe, стала очень простым занятием. В справочном материале, поставляемом с Visual Studio 2008, приводится много примеров использования этого класса. В листинге 16.6 приведен простой пример, который даст вам представление, как вы можете использовать данный класс в своем приложении.

**Листинг 16.6. Пинг адресов**

```
Ping pingSender = new Ping();

PingReply reply = pingSender.Send("rusproject.narod.ru");

if (reply.Status == IPStatus.Success)
 {
     listBox1.Items.Add("Address: " + reply.Address.ToString());
     listBox1.Items.Add("RoundTrip time: " + reply.RoundtripTime);
     listBox1.Items.Add("Time to live: " + reply.Options.Ttl);
     listBox1.Items.Add("Don't fragment: " + reply.Options.DontFragment);
     listBox1.Items.Add("Buffer size: " + reply.Buffer.Length);
 }
 else
```

```
{
   listBox1.Items.Add(reply.Status);
}
```

### *Примечание*

Пример работы с классом `Ping` находится в папке PingDemo на прилагаемом диске.

Также хочу предложить вашему вниманию альтернативный вариант, который был приведен программистом Младеном Янковичем (Mladen Jankovic) на сайте CodeProject по адресу **http://www.codeproject.com/KB/IP/SimplePingUtilityWithGUI.aspx**. Данный пример имеет GUI-интерфейс, и пользоваться этой программой гораздо удобнее. Проект немного сырой и требует доработки. Но на основе этого примера вы сможете создать свою собственную программу. Пример сопровождается хорошей статьей, где подробно описывается его работа.

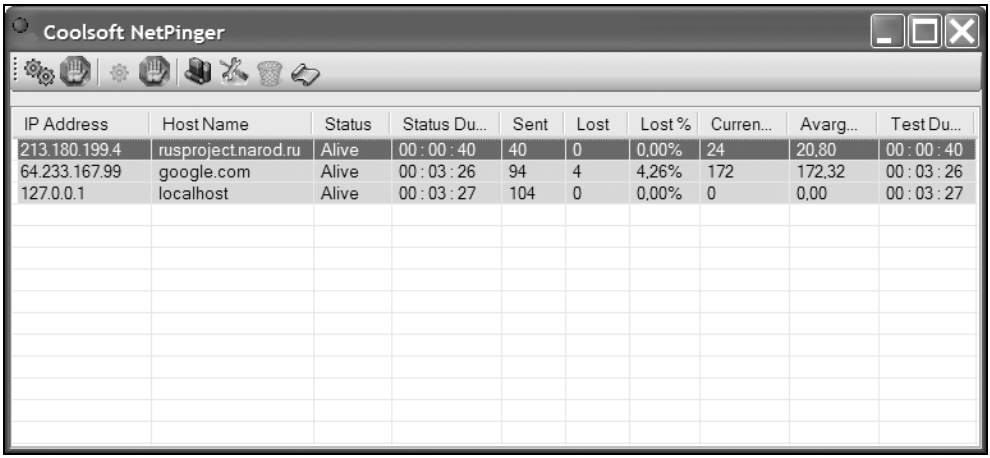



**Рис. 16.1.** Утилита для работы с классом `Ping` с графическим интерфейсом

### *Примечание*

Пример создания графического интерфейса для работы с классом `Ping` находится в папке NETPinger на прилагаемом диске.

## Проверка доступности веб-адреса

Если нужно определить, доступна ли указанная ссылка в Интернете, можно написать небольшую вспомогательную функцию `CheckURL`, приведенную в листинге 16.7.

**Листинг 16.7. Проверка доступности адреса в Интернете**

```
using System.Net;

public static bool CheckUrl(string url)
{
    bool rt = false;
    if (url.ToLower().StartsWith("www."))
    {
        url = "http://" + url;
    }

    HttpWebResponse myResponse = null;
    try
    {
        HttpWebRequest myRequest =
            (HttpWebRequest)WebRequest.Create(url);
        myResponse = (HttpWebResponse)myRequest.GetResponse();
        rt = true;
    }
    catch (WebException err)
    {
        rt = false;
    }
    finally
    {
        if (!(myResponse == null))
        {
            myResponse.Close();
        }
    }
    return rt;
}
```

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(CheckUrl(textBox1.Text).ToString());
}
```

Теперь вам достаточно ввести в текстовое поле какой-нибудь адрес и убедиться, что данный адрес доступен в сети. Можно использовать как название сайта (например, **www.stream.ru**), так и указывать конкретный файл на сайте (например, **www.whatis.ru/index.shtml**). В данном примере используется префикс www для проверки, что адрес является веб-сайтом. Можно сделать функцию более универсальной, чтобы она позволяла бы проверять доступность ссылки, не имеющей подобного префикса.

> ***ПРИМЕЧАНИЕ***
>
> Пример, в котором производится проверка адреса, находится в папке CheckUrlDemo на прилагаемом диске.

## Подключен ли компьютер к Интернету?

Чтобы проверить наличие подключения компьютера к Интернету, можно воспользоваться функцией Windows API `InternetGetConnectedState`, применение которой проиллюстрировано в листинге 16.8.

**Листинг 16.8. Проверка наличия подключения компьютера к Интернету**

```
// Функция Windows API
[DllImport("wininet.dll")]
static extern bool InternetGetConnectedState(
    ref InternetConnectionState  lpdwFlags, int dwReserved);

[Flags]
enum InternetConnectionState : int
{
    INTERNET_CONNECTION_MODEM = 0x1,
    INTERNET_CONNECTION_LAN = 0x2,
    INTERNET_CONNECTION_PROXY = 0x4,
    INTERNET_RAS_INSTALLED = 0x10,
    INTERNET_CONNECTION_OFFLINE = 0x20,
    INTERNET_CONNECTION_CONFIGURED = 0x40
}
```

```
private void button1_Click(object sender, EventArgs e)
{
    InternetConnectionState flags = 0;
    bool checkStatus = InternetGetConnectedState(ref flags, 0);
    MessageBox.Show(checkStatus.ToString());
}
```

# Пересылка данных по протоколу HTTP

Мы рассмотрим лишь два примера, которые дают общее представление об использовании протокола HTTP в .NET Framework. Для этой цели служит ряд классов.

## Как послать запрос GET и отобразить полученные данные?

В примере, приведенном в листинге 16.6, программа посылала запрос при помощи класса `HttpWebRequest` и обрабатывала полученный ответ (класс `HttpWebResponse`). С помощью комбинации "запрос-ответ" можно не только проверить доступность адреса, но и получить содержимое запрашиваемого адреса. Сделаем это (листинг 16.9) с применением похожих классов `WebRequest` и `WebResponse`.

**Листинг 16.9. Посыла запроса серверу и обработка ответа**

```
using System.Net;
using System.IO;

private void button1_Click(object sender, EventArgs e)
{
    string sURL;
    sURL = "http://netsources.narod.ru/";

    WebRequest wrGETURL;
    wrGETURL = WebRequest.Create(sURL);

    Stream objStream;
    objStream = wrGETURL.GetResponse().GetResponseStream();

    StreamReader objReader = new StreamReader(objStream);
```

```
    string sLine = "";
    int i = 0;

    while (sLine != null)
    {
        i++;
        sLine = objReader.ReadLine();
        if (sLine != null)
            textBox1.Text += string.Format("{0}:{1}", i, sLine);
    }
}
```

Как видите, с помощью .NET Framework очень просто сделать запрос GET для получения веб-страницы из Интернета, используя очень удобное пространство имен `System.Net`, что избавляет разработчика от необходимости использовать неуправляемые функции WinInet API. Классы `WebRequest` и `WebResponse` позволяют делать запрос интернет-ресурсов и обработать возвращаемые данные. Используя эти классы, можно обработать поток, получаемый после запроса, так же, как обычно это происходит при чтении текстового файла. Если вы используете прокси-сервер, то вам нужно использовать в приложении дополнительный код. Более подробную информацию на этот счет вы можете получить из статьи Базы Знаний "How to make a GET request by using Visual C#" по адресу **http://support.microsoft.com/kb/307023**/.

> ***Примечание***
>
> Пример обработки запроса находится в папке RequestResponse на прилагаемом диске.

## Как скачать файл из Интернета?

Скачать файл из Интернета, если вам известен его адрес, не составляет никакого труда. Достаточно воспользоваться классом `System.Net.WebClient` (листинг 16.10).

**Листинг 16.10. Закачка файла из Интернета**

```
using System.Net;

private void button2_Click(object sender, EventArgs e)
```

```
{
    string remoteUri = "http://rusproject.narod.ru/image/";
    string fileName = "ruspro.gif", myStringWebResource = null;

    // Создаем экземпляр класса WebClient
    WebClient myWebClient = new WebClient();

    // Соединим имя домена с именем файла
    myStringWebResource = remoteUri + fileName;

    textBox1.Text = "Загружаем файл " + fileName +
                    " из " + myStringWebResource + "\r\n";

    // Загрузим файл и сохраним его в локальной файловой системе
    myWebClient.DownloadFile(myStringWebResource, "c:\\" + fileName);
    textBox1.Text += "Файл " + fileName + " из " + myStringWebResource +
                     " успешно загружен" +"\r\n";
    textBox1.Text += "Загруженный файл сохранен на диске C:";
}
```

***Примечание***

Примеры проверки подключения и загрузки файла находятся в папке WinInet на прилагаемом диске.

# Передача файлов по протоколу FTP

В .NET Framework 2.0 появились классы, позволяющие разрабатывать приложения, которые могут использовать протокол FTP. До этого приходилось использовать неуправляемые функции Windows API. Я расскажу об основных приемах, используемых в программах такого рода. Для начала надо объявить необходимые пространства имен:

```
using System.NET;
using System.IO;
```

Пространство имен `System.NET` необходимо для подключения классов, работающих с протоколом FTP, а пространство имен `System.IO` позволит нам проводить необходимые манипуляции с файлами. Далее надо придерживаться следующей последовательности:

1. Создать объект `FtpWebRequest`, который позволит подключиться к FTP-серверу.

2. Установить в свойстве `Method` созданного объекта необходимый тип операции с FTP-сервером (закачка файла на сервер, скачивание файла с сервера, создание папки и т. п.).
3. Сделать требуемые настройки соединения (поддержка SSL, бинарный режим передачи файлов и т. д.).
4. Указать данные для подключения к FTP-серверу (логин и пароль).
5. Выполнить указанную команду.
6. Получить поток (если требуется).
7. Закрыть запрос и открытые потоки.

Теперь приведем несколько простых примеров, которые наиболее часто используются при соединении с FTP-сервером.

## Закачка файла на FTP-сервер

Сначала мы закачаем файл на FTP-сервер. Для примера нам понадобится существующий сервер, с которым можно соединиться. Если у вас нет такого сервера, то можете завести себе собственную страницу на **http://narod.yandex.ru**, и у вас будет возможность закачивать файлы на сервер **ftp.narod.ru**, используя логин и пароль, полученный при регистрации. Именно к этому серверу мы и обращаемся в листинге 16.11.

**Листинг 16.11. Закачка файла на FTP-сервер**

```
using System.Net;
using System.IO;

// Закачка файла на FTP-сервер
private void FTPUploadFile(string filename)
{

    FileInfo fileInfo = new FileInfo(filename);

    string uri = "ftp://" + "ftp.narod.ru" + "/" + fileInfo.Name;

    FtpWebRequest reqFTP;

    // Создаем объект FtpWebRequest, используя заданный адрес
```

```
reqFTP = (FtpWebRequest)FtpWebRequest.Create(
    new Uri("ftp://" + "ftp.narod.ru" + "/" + fileInfo.Name));

// Используем учетную запись для доступа
// Используйте реальные данные
// В этом примере используются вымышленные данные
reqFTP.Credentials = new NetworkCredential("csharp", "narod");

// По умолчанию свойство KeepAlive равно true,
// если соединение не обрывается после выполнения команды
reqFTP.KeepAlive = false;

// Задаем нужную команду.
reqFTP.Method = WebRequestMethods.Ftp.UploadFile;

// Указываем тип данных при передаче файлов
reqFTP.UseBinary = true;

// Сообщаем серверу о размере закачиваемого файла
reqFTP.ContentLength = fileInfo.Length;

// Устанавливаем размер буфера в 2 Кбайт
int buffLength = 2048;
byte[] buff = new byte[buffLength];
int contentLen;

// Открываем файловый поток (System.IO.FileStream) для чтения
FileStream fs = fileInfo.OpenRead();

try
{
    // Поток, в который записывается закачиваемый на сервер файл
    Stream strm = reqFTP.GetRequestStream();

    // Читаем из файлового потока по 2 Кбайт за раз
    contentLen = fs.Read(buff, 0, buffLength);

    // Пока файл не закончился
    while (contentLen != 0)
    {
        // Запишем прочитанное в поток закачки
        strm.Write(buff, 0, contentLen);
```

```
            contentLen = fs.Read(buff, 0, buffLength);
        }

        // Закрываем файловый поток и поток запроса
        strm.Close();
        fs.Close();
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка при закачке файла");
    }
}


private void button1_Click(object sender, EventArgs e)
{
    FTPUploadFile(@"c:\ruspro.gif");
}
```

## Получение оглавления папки

Мы рассмотрели пример закачки файла. Предположим, что теперь мы хотим, наоборот, скачать файл с FTP-сервера к себе на жесткий диск. Для этого нам необходимо получить список файлов и папок, расположенных на сервере. Пример, приведенный в листинге 16.12, демонстрирует, как получить такой список файлов.

**Листинг 16.12. Получение списка файлов**

```
public string[] GetFileList()
{
    string[] downloadFiles;
    StringBuilder result = new StringBuilder();
    FtpWebRequest reqFTP;
    try
    {
        reqFTP = (FtpWebRequest)FtpWebRequest.Create(
                    new Uri("ftp://" + "ftp.narod.ru" + "/"));
        reqFTP.UseBinary = true;
        // Используйте реальные данные пользователя
        reqFTP.Credentials = new NetworkCredential("csharp", "narod");
```

```
        // Задаем команду получения списка файлов
        reqFTP.Method = WebRequestMethods.Ftp.ListDirectory;

        WebResponse response = reqFTP.GetResponse();
        StreamReader reader =
            new StreamReader(response.GetResponseStream());

        string line = reader.ReadLine();

        while (line != null)
        {
            result.Append(line);
            result.Append("\n");
            line = reader.ReadLine();
        }

        // Удаляем завершающие символы '\n'
        result.Remove(result.ToString().LastIndexOf('\n'), 1);
        reader.Close();
        response.Close();
        return result.ToString().Split('\n');
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        downloadFiles = null;
        return downloadFiles;
    }
}

private void button2_Click(object sender, EventArgs e)
{
    // Выводим список файлов
    string[] allFiles = GetFileList();
    foreach (string file in allFiles)
    {
        listBox1.Items.Add(file);
    }
}
```

## Загрузка файлов

Мы научились закачивать файл на FTP-сервер, а также просматривать структуру файлов на этом сервере. Теперь наша задача — загрузить какой-нибудь файл с сервера на свой компьютер. Она решается в листинге 16.13.

**Листинг 16.13. Загрузка файла на локальный компьютер**

```
private void FTPDownloadFile(string filePath, string fileName)
{
    FtpWebRequest reqFTP;
    try
    {
        //filePath — полный путь к папке, где должен быть создан файл
        //fileName - имя создаваемого файла на локальном компьютере

        FileStream outputStream =
            new FileStream(filePath + "\\" + fileName, FileMode.Create);

        reqFTP = (FtpWebRequest)FtpWebRequest.Create(new
           Uri("ftp://" + "ftp.narod.ru" + "/" + fileName));
        reqFTP.Method = WebRequestMethods.Ftp.DownloadFile;
        reqFTP.UseBinary = true;

        // Используйте реальные данные для логина и пароля
        reqFTP.Credentials = new NetworkCredential("csharp", "narod");
        FtpWebResponse response = (FtpWebResponse)reqFTP.GetResponse();

        Stream ftpStream = response.GetResponseStream();
        long cl = response.ContentLength;
        int bufferSize = 2048;
        int readCount;
        byte[] buffer = new byte[bufferSize];

        readCount = ftpStream.Read(buffer, 0, bufferSize);
        while (readCount > 0)
        {
            outputStream.Write(buffer, 0, readCount);
            readCount = ftpStream.Read(buffer, 0, bufferSize);
        }
```

```
        ftpStream.Close();
        outputStream.Close();
        response.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
private void button3_Click(object sender, EventArgs e)
{
    FTPDownloadFile("c:/temp", "ruspro.gif");
}
```

***Примечание***

Пример работы с FTP находится в папке FTPDemo на прилагаемом диске.

# Отправка писем через SMTP

В .NET Framework 2.0 также появилось новое пространство имен `System.Net.Mail`, которое содержит классы, используемые для отправки электронных писем при помощи сервера SMTP. Для работы с этим пространством имен необходимо добавить также ссылку на пространство имен `System.Net`. Как им пользоваться, показано в листинге 16.14.

**Листинг 16.14. Посылаем письмо через SMTP**

```
// Отправим письмо через почтовый сервер mail.ru
using System.Net.Mail;
using System.Net;

private void butSend_Click(object sender, EventArgs e)
{
    MailAddress from = new MailAddress("alibaba@mail.ru", "Ali Baba");
    MailAddress to = new MailAddress("alladin@rambler.ru");

    MailMessage message = new MailMessage(from, to);
    message.Subject = txtSubject.Text;
    message.Body = txtBody.Text;
```

```
    // Можете добавить скрытую копию
    //MailAddress copy = new MailAddress("cat@cat.com");
    //message.CC.Add(copy);

    SmtpClient client = new SmtpClient("smtp.mail.ru");
    client.Timeout = 2000;

    // Включаем проверку подлинности пользователя,
    // если этого требует сервер. 2 способа:
    // 1 способ
    client.Credentials = CredentialCache.DefaultNetworkCredentials;
    // 2 способ - можно использовать настройки по умолчанию
    //client.UseDefaultCredentials = true;

    // Посылаем письмо
    client.Send(message);
}
```

***ПРИМЕЧАНИЕ***

Пример отправки письма находится в папке SMTPMail на прилагаемом диске.

# Использование браузера Mozilla Firefox

Многие пользователи используют для просмотра веб-страниц альтернативные браузеры: Opera, Mozilla Firefox и др. Сергей Борзов на странице своего блога **http://seregaborzov.wordpress.com** разместил пример класса, который взаимодействует с браузером Mozilla Firefox. С небольшими изменениями я предлагаю этот пример вашему вниманию (листинг 16.15).

**Листинг 16.15. Использование Mozilla Firefox**

```
class Firefox
{
     private string firefoxPath;

     /// <summary>
     /// Конструктор класса Firefox.
     /// </summary>
     public Firefox()
```

```
{
    firefoxPath =
        string.Format(@"{0}\Mozilla Firefox\firefox.exe",
        Environment.GetFolderPath(
            Environment.SpecialFolder.ProgramFiles));
}

/// <summary>
/// Открывает Firefox, если он доступен
/// </summary>
/// <param name="arguments">веб-адрес (url)</param>
private void Open(string arguments)
{
    if (!IsFirefoxAvailable())
        throw new Exception("Firefox не установлен.");
    else
        Process.Start(firefoxPath, arguments);
}

/// <summary>
/// Определяет наличие установленного Firefox.
/// </summary>
/// <returns>
/// true если Firefox установлен; false - если не установлен.
/// </returns>
public bool IsFirefoxAvailable()
{
    FileInfo fiFirefox = new FileInfo(firefoxPath);
    return fiFirefox.Exists;
}

/// <summary>
/// Запускает Firefox.
/// </summary>
public void OpenFirefox()
{
    Open(string.Empty);
}

/// <summary>
/// Запускает Firefox с заданнам url.
/// </summary>
```

```
    /// <param name="url">url</param>
    public void OpenFirefox(string url)
    {
        Open(url);
    }

    /// <summary>
    /// Запускает Firefox в безопасном режиме (save mode).
    /// </summary>
    public void OpenFirefoxInSaveMode()
    {
        Open("-safe-mode");
    }

    /// <summary>
    /// Открывает Firefox в новом окне.
    /// </summary>
    /// <param name="url">url</param>
    public void OpenFirefoxInNewWindow(string url)
    {
        Open(string.Format("-new-window {0}", url));
    }

    /// <summary>
    /// Открывает Firefox в новой вкладке.
    /// </summary>
    /// <param name="url">url</param>
    public void OpenFirefoxInNewTab(string url)
    {
        Open(string.Format("-new-tab {0}", url));
    }

    /// <summary>
    /// Открывает русскую страницу Firefox.
    /// </summary>
    public void OpenMozillaWebsite()
    {
        Process.Start("http://www.mozilla.ru");
    }
}
```

После создания класса можно использовать методы класса `Firefox` в своих программах, например, как показано в листинге 16.16.

**Листинг 16.16. Использование класса `Firefox` в проекте**

```
public partial class Form1 : Form
{
    private Firefox fox;

    public Form1()
    {
        InitializeComponent();
        fox = new Firefox();
    }

    private void butOpenFirefox_Click(object sender, EventArgs e)
    {
        fox.OpenFirefox();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        fox.OpenFirefox("http://www.bhv.ru");
    }

    private void butTab_Click(object sender, EventArgs e)
    {
        fox.OpenFirefoxInNewTab("http://netsources.narod.ru");

    }

    private void butMozillaSite_Click(object sender, EventArgs e)
    {
        fox.OpenMozillaWebsite();
    }
}
```

***Примечание***

Пример находится в папке FirefoxSample на прилагаемом диске.

# Работа с локальной сетью

Не менее важным аспектом, чем программирование сети Интернет, является программирование локальной сети. Мы рассмотрим несколько приемов.

## Как получить имя текущего пользователя?

Класс `System.Environment` содержит свойства `UserDomainName` и `UserName`, с помощью которых можно получить информацию о текущем пользователе системы. Свойство `UseDomainName` возвращает имя сетевого домена, к которому подключен текущий пользователь, а свойство `UserName` — имя находящегося в системе пользователя, который запустил текущий поток команд. Комбинация из этих двух свойств позволят нам идентифицировать пользователя (листинг 16.17).

**Листинг 16.17. Получение имени текущего пользователя**

```
private void butGetUserInfo_Click(object sender, EventArgs e)
{
    MessageBox.Show(Environment.UserDomainName + @"\" +
        Environment.UserName);
}
```

Есть еще и второй способ с использованием пространства имен `System.Security.Principal`, который позволяет получить такую же информацию о текущем пользователе (листинг 16.18).

**Листинг 16.18. Второй способ получения имени текущего пользователя**

```
using System.Security.Principal;

private void butGetUserInfo2_Click(object sender, EventArgs e)
{
    WindowsIdentity user = WindowsIdentity.GetCurrent();
    MessageBox.Show(user.Name.ToString());
}
```

***Примечание***

Пример получения информации о текущем пользователе находится в папке CurrentUser на прилагаемом диске.

# Как выяснить, подключена ли локальная система к сети, и узнать используемый тип соединения?

Сетевая функция Windows API `IsNetworkAlive` определяет, подключена ли локальная система к сети, а также возвращает информацию о типе соединения (LAN, WAN). Пример ее использования показан в листинге 16.19.

**Листинг 16.19. Проверка подключения локальной системы к сети**

```
[DllImport("sensapi.dll")]
private extern static bool IsNetworkAlive(ref int flags);

private static int NETWORK_ALIVE_LAN = 0x00000001;
private static int NETWORK_ALIVE_WAN = 0x00000002;

public static bool IsLanAlive()
{
    return IsNetworkAlive(ref NETWORK_ALIVE_LAN);
}
public static bool IsWanAlive()
{
    return IsNetworkAlive(ref NETWORK_ALIVE_WAN);
}

private void button1_Click(object sender, EventArgs e)
{
    this.Text = IsLanAlive().ToString();
}

private void button2_Click(object sender, EventArgs e)
{
    this.Text = IsWanAlive().ToString();
}
```

***Примечание***

Пример проверки связи находится в папке NetworkDemo на прилагаемом диске.

# Получение списка всех компьютеров локальной сети

Если вы работаете в локальной сети, то, несомненно, для вас интересен вопрос, как получить список всех работающих компьютеров в локальной сети или, например, список серверов SQL. Также хочется знать, как получить имя домена, контроллера домена и другую информацию. Встроенными средствами .NET Framework реализовать эту задачу пока не удастся. В листинге 16.20 показано, как достичь результата, воспользовавшись функциями Windows API.

**Листинг 16.20. Получение списка всех компьютеров локальной сети**

```
using System.Runtime.InteropServices;
using System.Collections;

[DllImport("netapi32.dll", EntryPoint = "NetServerEnum")]
public static extern NERR NetServerEnum(
     [MarshalAs(UnmanagedType.LPWStr)]string ServerName,
     int Level, out IntPtr BufPtr,
     int PrefMaxLen, ref int EntriesRead,
     ref int TotalEntries, SV_101_TYPES ServerType,
     [MarshalAs(UnmanagedType.LPWStr)] string Domain,
     int ResumeHandle);

[DllImport("netapi32.dll", EntryPoint = "NetApiBufferFree")]
public static extern NERR NetApiBufferFree(IntPtr Buffer);

[StructLayout(LayoutKind.Sequential)]
public struct SERVER_INFO_101
{
    [MarshalAs(UnmanagedType.U4)]
    public uint sv101_platform_id;
    [MarshalAs(UnmanagedType.LPWStr)]
    public string sv101_name;
    [MarshalAs(UnmanagedType.U4)]
    public uint sv101_version_major;
    [MarshalAs(UnmanagedType.U4)]
    public uint sv101_version_minor;
    [MarshalAs(UnmanagedType.U4)]
    public uint sv101_type;
```

```
    [MarshalAs(UnmanagedType.LPWStr)]
    public string sv101_comment;
}

/// <summary>
/// Список ошибок, возвращаемых NetServerEnum
/// </summary>
public enum NERR
{
    NERR_Success = 0, // успех
    ERROR_ACCESS_DENIED = 5,
    ERROR_NOT_ENOUGH_MEMORY = 8,
    ERROR_BAD_NETPATH = 53,
    ERROR_NETWORK_BUSY = 54,
    ERROR_INVALID_PARAMETER = 87,
    ERROR_INVALID_LEVEL = 124,
    ERROR_MORE_DATA = 234,
    ERROR_EXTENDED_ERROR = 1208,
    ERROR_NO_NETWORK = 1222,
    ERROR_INVALID_HANDLE_STATE = 1609,
    ERROR_NO_BROWSER_SERVERS_FOUND = 6118,
}

/// <summary>
/// Типы серверов
/// </summary>
[Flags]
public enum SV_101_TYPES : uint
{
    SV_TYPE_WORKSTATION = 0x00000001,
    SV_TYPE_SERVER = 0x00000002,
    SV_TYPE_SQLSERVER = 0x00000004,
    SV_TYPE_DOMAIN_CTRL = 0x00000008,
    SV_TYPE_DOMAIN_BAKCTRL = 0x00000010,
    SV_TYPE_TIME_SOURCE = 0x00000020,
    SV_TYPE_AFP = 0x00000040,
    SV_TYPE_NOVELL = 0x00000080,
    SV_TYPE_DOMAIN_MEMBER = 0x00000100,
    SV_TYPE_PRINTQ_SERVER = 0x00000200,
    SV_TYPE_DIALIN_SERVER = 0x00000400,
    SV_TYPE_XENIX_SERVER = 0x00000800,
    SV_TYPE_SERVER_UNIX = SV_TYPE_XENIX_SERVER,
```

```
    SV_TYPE_NT = 0x00001000,
    SV_TYPE_WFW = 0x00002000,
    SV_TYPE_SERVER_MFPN = 0x00004000,
    SV_TYPE_SERVER_NT = 0x00008000,
    SV_TYPE_POTENTIAL_BROWSER = 0x00010000,
    SV_TYPE_BACKUP_BROWSER = 0x00020000,
    SV_TYPE_MASTER_BROWSER = 0x00040000,
    SV_TYPE_DOMAIN_MASTER = 0x00080000,
    SV_TYPE_SERVER_OSF = 0x00100000,
    SV_TYPE_SERVER_VMS = 0x00200000,
    SV_TYPE_WINDOWS = 0x00400000,
    SV_TYPE_DFS = 0x00800000,
    SV_TYPE_CLUSTER_NT = 0x01000000,
    SV_TYPE_TERMINALSERVER = 0x02000000,
    SV_TYPE_CLUSTER_VS_NT = 0x04000000,
    SV_TYPE_DCE = 0x10000000,
    SV_TYPE_ALTERNATE_XPORT = 0x20000000,
    SV_TYPE_LOCAL_LIST_ONLY = 0x40000000,
    SV_TYPE_DOMAIN_ENUM = 0x80000000,
    SV_TYPE_ALL = 0xFFFFFFFF,
}

// Получить список SQL-серверов
public static ArrayList GetSQLServerList()
{
    SERVER_INFO_101 si;
    IntPtr pInfo = IntPtr.Zero;
    int etriesread = 0;
    int totalentries = 0;
    ArrayList srvs = new ArrayList();
    try
    {
        NERR err = NetServerEnum(null, 101, out pInfo, -1,
             ref etriesread, ref totalentries,
             SV_101_TYPES.SV_TYPE_SQLSERVER, null, 0);
        if ((err == NERR.NERR_Success ||
                     err == NERR.ERROR_MORE_DATA) &&
                     pInfo != IntPtr.Zero)
        {
            int ptr = pInfo.ToInt32();
            for (int i = 0; i < etriesread; i++)
```

```
            {
                si = (SERVER_INFO_101)Marshal.PtrToStructure(
                       new IntPtr(ptr), typeof(SERVER_INFO_101));
                srvs.Add(si.sv101_name); // Добавляем имя
                                         // сервера в список
                ptr += Marshal.SizeOf(si);
            }
        }
    }
    catch (Exception)
    {
    }
    finally
    { // Освобождаем выделенную память
        if (pInfo != IntPtr.Zero)
        {
            NetApiBufferFree(pInfo);
        }
    }
    return (srvs);
}

// Получить список всех компьютеров
public static ArrayList GetServerList(SV_101_TYPES type)
{
    SERVER_INFO_101 si;
    IntPtr pInfo = IntPtr.Zero;
    int etriesread = 0;
    int totalentries = 0;
    ArrayList srvs = new ArrayList();
    try
    {
        NERR err = NetServerEnum(null, 101, out pInfo, -1,
             ref etriesread, ref totalentries,
             SV_101_TYPES. SV_TYPE_ALL, null, 0);
        if ((err == NERR.NERR_Success || err == NERR.ERROR_MORE_DATA)
             && pInfo != IntPtr.Zero)
        {
            int ptr = pInfo.ToInt32();
            for (int i = 0; i < etriesread; i++)
            {
                si = (SERVER_INFO_101)Marshal.PtrToStructure(
```

```
                        new IntPtr(ptr), typeof(SERVER_INFO_101));
                srvs.Add(si.sv101_name); // Добавляем имя сервера
                                         // в список
                ptr += Marshal.SizeOf(si);
            }
        }
    }
    catch (Exception) { /* Обработка ошибки */ }
    finally
    { // Освобождаем выделенную память
        if (pInfo != IntPtr.Zero) NetApiBufferFree(pInfo);
    }
    return (srvs);
}

public static ArrayList GetDomenControllerList(SV_101_TYPES type)
{
    SERVER_INFO_101 si;
    IntPtr pInfo = IntPtr.Zero;
    int etriesread = 0;
    int totalentries = 0;
    ArrayList srvs = new ArrayList();
    try
    {
        NERR err = NetServerEnum(null, 101, out pInfo, -1,
             ref etriesread, ref totalentries,
             SV_101_TYPES.SV_TYPE_DOMAIN_CTRL, null, 0);
        if ((err == NERR.NERR_Success || err == NERR.ERROR_MORE_DATA)
            && pInfo != IntPtr.Zero)
        {
            int ptr = pInfo.ToInt32();
            for (int i = 0; i < etriesread; i++)
            {
                si = (SERVER_INFO_101)Marshal.PtrToStructure(
                         new IntPtr(ptr), typeof(SERVER_INFO_101));
                srvs.Add(si.sv101_name); // Добавляем имя сервера
                                         // в список
                ptr += Marshal.SizeOf(si);
            }
        }
    }
```

```
    catch (Exception) { /* Обработка ошибки */ }
    finally
    { // Освобождаем выделенную память
        if (pInfo != IntPtr.Zero) NetApiBufferFree(pInfo);
    }
    return (srvs);
}

// Получить список доменов
public static ArrayList GetDomenList()
{
    SERVER_INFO_101 si;
    IntPtr pInfo = IntPtr.Zero;
    int etriesread = 0;
    int totalentries = 0;
    ArrayList srvs = new ArrayList();
    try
    {
        NERR err = NetServerEnum(null, 101, out pInfo, -1,
             ref etriesread, ref totalentries,
             SV_101_TYPES.SV_TYPE_DOMAIN_ENUM, null, 0);
        if ((err == NERR.NERR_Success || err == NERR.ERROR_MORE_DATA)
             && pInfo != IntPtr.
        {
            int ptr = pInfo.ToInt32();
            for (int i = 0; i < etriesread; i++)
            {
                si = (SERVER_INFO_101)Marshal.PtrToStructure(
                         new IntPtr(ptr), typeof(SERVER_INFO_101));
                srvs.Add(si.sv101_name); // Добавляем имя сервера
                                         // в список
                ptr += Marshal.SizeOf(si);
            }
        }
    }
    catch (Exception) { /* Обработка ошибки */ }
    finally
    { // Освобождаем выделенную память
        if (pInfo != IntPtr.Zero) NetApiBufferFree(pInfo);
    }
    return (srvs);
}
```

```
// Получить список доменов
public static ArrayList GetDomenList()
{
    SERVER_INFO_101 si;
    IntPtr pInfo = IntPtr.Zero;
    int etriesread = 0;
    int totalentries = 0;
    ArrayList srvs = new ArrayList();
    try
    {
        NERR err = NetServerEnum(null, 101, out pInfo, -1,
             ref etriesread, ref totalentries,
             SV_101_TYPES.SV_TYPE_DOMAIN_ENUM, null, 0);
        if ((err == NERR.NERR_Success || err == NERR.ERROR_MORE_DATA)
              && pInfo != IntPtr.Zero)
        {
            int ptr = pInfo.ToInt32();
            for (int i = 0; i < etriesread; i++)
            {
                si = (SERVER_INFO_101)Marshal.PtrToStructure(
                         new IntPtr(ptr), typeof(SERVER_INFO_101));
                srvs.Add(si.sv101_name); // Добавляем имя сервера
                                         // в список
                ptr += Marshal.SizeOf(si);
            }
        }
    }
    catch (Exception) { /* обработка ошибки */ }
    finally
    { // освобождаем выделенную память
        if (pInfo != IntPtr.Zero) NetApiBufferFree(pInfo);
    }
    return (srvs);
}

private void button1_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();

    ArrayList list = GetSQLServerList();
    foreach (string name in list)
    {
```

```
        listBox1.Items.Add(name);
    }
}

private void button2_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();

    ArrayList list = GetServerList(SV_101_TYPES.SV_TYPE_ALL);
    foreach (string name in list)
    {
        listBox1.Items.Add(name);
    }
}

private void button3_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();

    ArrayList list =
        GetDomenControllerList(SV_101_TYPES.SV_TYPE_DOMAIN_CTRL);
    foreach (string name in list)
    {
        listBox1.Items.Add(name);
    }
}

private void button4_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();

    ArrayList list = GetDomenList();
    foreach (string name in list)
    {
        listBox1.Items.Add(name);
    }
}
```

### *Примечание*

Пример получения списка компьютеров в сети находится в папке ListComputersLAN на прилагаемом диске.

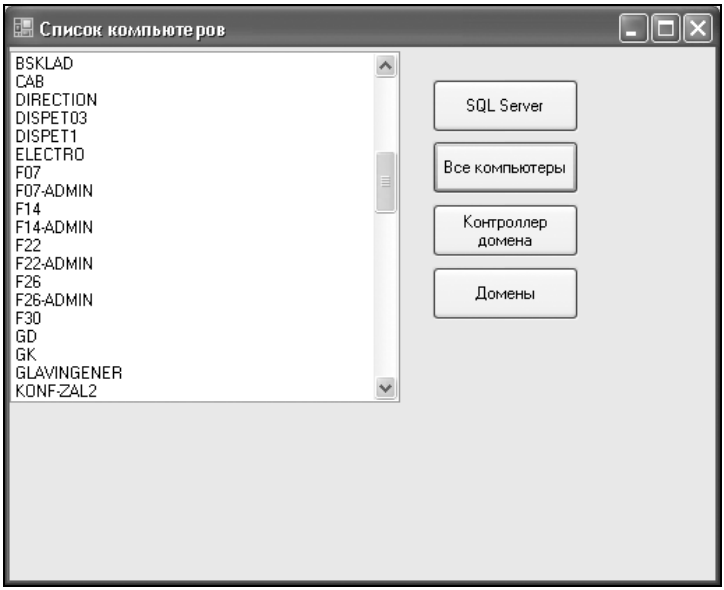



**Рис. 16.2.** Список всех компьютеров в сети

## Список SQL-серверов при помощи управляемого кода

В предыдущем примере, чтобы получить список SQL-серверов, мы использовали неуправляемый код. Но существует еще один альтернативный вариант с использованием встроенных классов .NET Framework (листинг 16.21).

**Листинг 16.21. Список SQL-серверов в сети при помощи управляемого кода**

```
using System.Data.Sql;
using System;

class Program
{
    static void Main()
    {
        // Получаем перечеслитель, а затем данные
        SqlDataSourceEnumerator instance =
```

```
            SqlDataSourceEnumerator.Instance;
        System.Data.DataTable table = instance.GetDataSources();

        // Отображаем содержимое таблицы
        DisplayData(table);

        Console.WriteLine("Нажмите любую клавишу, чтобы продолжить");
        Console.ReadKey();
    }

    private static void DisplayData(System.Data.DataTable table)
    {
        foreach (System.Data.DataRow row in table.Rows)
        {
            foreach (System.Data.DataColumn col in table.Columns)
            {
                Console.WriteLine("{0} = {1}", col.ColumnName, row[col]);
            }
            Console.WriteLine("============================");
        }

    }
}
```

Этот пример консольного приложения описан в документации MSDN. Поэтому никогда не забывайте заглянуть в документацию в поисках решения своих проблем.

> ***Примечание***
>
> Пример получения списка SQL-серверов находится в папке SQLEnum на прилагаемом диске.

## Как получить дату и время удаленного компьютера?

Если вам необходимо получить дату и время на удаленном компьютере в сети, то воспользуйтесь вызовом функции Windows API `NetRemoteTOD`, как показано в листинге 16.22.

**Листинг 16.22. Получение даты и времени удаленного компьютера**

```
using System.Runtime.InteropServices;

[DllImport("netapi32.dll", EntryPoint = "NetRemoteTOD",
SetLastError = true, CharSet = CharSet.Unicode, ExactSpelling = true,
CallingConvention = CallingConvention.StdCall)]
private static extern int NetRemoteTOD(string UncServerName,
ref IntPtr BufferPtr);

[DllImport("netapi32.dll", EntryPoint = "NetApiBufferFree")]
public static extern NERR NetApiBufferFree(IntPtr Buffer);

[StructLayout(LayoutKind.Sequential)]
private struct TIME_OF_DAY_INFO
{
    public int tod_elapsedt;
    public int tod_msecs;
    public int tod_hours;
    public int tod_mins;
    public int tod_secs;
    public int tod_hunds;
    public int tod_timezone;
    public int tod_tinterval;
    public int tod_day;
    public int tod_month;
    public int tod_year;
    public int tod_weekday;
}

// список ошибок, возвращаемых NetServerEnum
public enum NERR
{
    NERR_Success = 0, // успех
    ERROR_ACCESS_DENIED = 5,
    ERROR_NOT_ENOUGH_MEMORY = 8,
    ERROR_BAD_NETPATH = 53,
    ERROR_NETWORK_BUSY = 54,
    ERROR_INVALID_PARAMETER = 87,
    ERROR_INVALID_LEVEL = 124,
    ERROR_MORE_DATA = 234,
    ERROR_EXTENDED_ERROR = 1208,
```

```
    ERROR_NO_NETWORK = 1222,
    ERROR_INVALID_HANDLE_STATE = 1609,
    ERROR_NO_BROWSER_SERVERS_FOUND = 6118,
}

public int[] GetRemoteTime(string ServerName)
{
    TIME_OF_DAY_INFO result = new TIME_OF_DAY_INFO();
    IntPtr pintBuffer = IntPtr.Zero;
    int[] TOD_INFO = new int[12];

    // Получим дату и время с сервера
    int pintError = NetRemoteTOD(ServerName, ref pintBuffer);
    if (pintError > 0) { throw
              new System.ComponentModel.Win32Exception(pintError); }

    // Получим данные структуры
    result = (TIME_OF_DAY_INFO)Marshal.PtrToStructure(
                                  pintBuffer, typeof(TIME_OF_DAY_INFO));

    TOD_INFO[0] = result.tod_elapsedt;
    TOD_INFO[1] = result.tod_msecs;
    TOD_INFO[2] = result.tod_hours;
    TOD_INFO[3] = result.tod_mins;
    TOD_INFO[4] = result.tod_secs;
    TOD_INFO[5] = result.tod_hunds;
    TOD_INFO[6] = result.tod_timezone;
    TOD_INFO[7] = result.tod_tinterval;
    TOD_INFO[8] = result.tod_day;
    TOD_INFO[9] = result.tod_month;
    TOD_INFO[10] = result.tod_year;
    TOD_INFO[11] = result.tod_weekday;

    // Освобождаем буфер
    NetApiBufferFree(pintBuffer);

    return TOD_INFO;
}

private void button1_Click(object sender, EventArgs e)
{
    string ServerName = @"\\skynet";
```

```
    int[] TOD_Info = new int[12];

    try
    {
        TOD_Info = GetRemoteTime(ServerName);

    }
    catch (Exception err)
    {
        // Выводим сообщение об ошибке
        MessageBox.Show("NetRemoteTOD from " + ServerName
            + " failed!\nError: " + err.Message);
        return;
    }

    TOD_Info[2] -= TOD_Info[6] / 60;

    MessageBox.Show(String.Format(
            "Дата/Время сервера {0} : {1}.{2}.{3}  {4}:{5}:{6}",
            ServerName, TOD_Info[8], TOD_Info[9],
            TOD_Info[10], TOD_Info[2], TOD_Info[3],
            TOD_Info[4]));
}
```

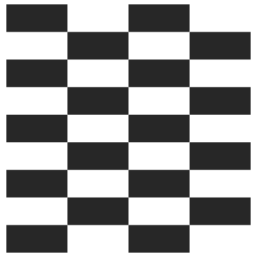
***ПРИМЕЧАНИЕ***

Пример получения даты с сервера находится в папке NetRemoteTODSample на прилагаемом диске.

# Заключение

В связи с бурным развитием Интернета и локальных сетей знание протоколов связи между компьютерами становится очень важным аспектом деятельности программиста.

## Глава 17



# Функции Windows API

Программный код, который выполняется под управлением CLR (Common Language Runtime, то есть общая среда выполнения языков), называется управляемым (managed) кодом. Программный код, выполняющийся вне среды выполнения CLR, называется неуправляемым (unmanaged) кодом. Примером неуправляемого программного кода служат функции Win32 API, компоненты COM, интерфейсы ActiveX. Несмотря на большое количество классов .NET Framework, содержащих множество методов, программисту все равно приходится иногда прибегать к неуправляемому коду. Надо сказать, что число вызовов неуправляемого кода уменьшается с выходом каждой новой версии .NET Framework. Microsoft надеется, что наступит такое время, когда весь код можно будет сделать управляемым и безопасным. Но пока реальность такова, что без вызовов функций Windows API не обойтись, что вы могли наблюдать на примерах из предыдущих глав. Здесь мы приведем еще несколько полезных приемов с использованием функций Windows API, которые не попали в предыдущие главы. Но сначала немного теории.

Управляемый код .NET Framework может вызывать неуправляемую функцию из DLL (функцию Windows API) при помощи специального механизма Platform Invoke (сокр. P/Invoke). Для того чтобы обратиться к какой-нибудь неуправляемой библиотеке DLL, вы должны преобразовать .NET-объекты в наборы struct, char* и указателей на функции, как того требует язык C. Как сказали бы программисты на своем жаргоне — вам нужно маршалировать параметры. Более подробно о маршалинге (Marshalling) вам следует почитать в документации. Чтобы вызвать DLL-функцию из C#, сначала ее необходимо объявить (программисты, имеющие опыт работы с Visual Basic 6.0, уже знакомы с этим способом). Для этого используется атрибут `DllImport`, например, как показано в листинге 17.1.

**Листинг 17.1. Объявление и использование функции Windows API**

```
using System.Runtime.InteropServices;

public class Win32
{
    [DllImport("User32.Dll")]
    public static extern void SetWindowText(IntPtr hwnd,
        String lpString);
}
```

Атрибут `DllImport` сообщает компилятору, где находится точка входа, что позволяет далее вызывать функцию из нужного места. Вы должны всегда использовать тип `IntPtr` для `HWND`, `HMENU` и любых других описателей. Для `LPCTSTR` используйте `String`, а сервисы взаимодействия (interop services) выполнят автоматический маршалинг `System.String` в `LPCTSTR` до передачи в Windows. Компилятор ищет указанную выше функцию `SetWindowText` в файле User32.dll и перед ее вызовом автоматически преобразует вашу строку в `LPTSTR` (`TCHAR*`). Почему это происходит? Для каждого типа в C# определен свой тип, используемый при маршалинге по умолчанию (default marshaling type) . Для строк это `LPTSTR`.

Часть советов, приводимых ниже, были взяты из разных номеров журнала "MSDN Magazine".

# Вызов функций Windows API, имеющих выходной строковый параметр *char**

Предположим, нам необходимо вызвать функцию `GetWindowText`, у которой имеется строковый выходной параметр `char*`. По умолчанию для строк используется `LPTSTR`, но если мы будем использовать `System.String`, как было сказано выше, то ничего не произойдет, так как класс `System.String` не позволяет модифицировать строку. Вам необходимо использовать класс `StringBuilder`, который позволяет изменять строки. Пример показан в листинге 17.2.

**Листинг 17.2. Работа с строковыми параметрами в функциях Windows API**

```
using System.Text;  // для StringBuilder

[DllImport("user32.dll")]
public static extern int GetWindowText(IntPtr hwnd,
StringBuilder buf, int nMaxCount);
```

Тип, используемый для маршалинга `StringBuilder` по умолчанию, — тоже `LPTSTR`, зато теперь `GetWindowText` может модифицировать саму вашу строку (листинг 17.3).

**Листинг 17.3. Применение функции `GetWindowText`**

```
StringBuilder sTitleBar = new StringBuilder(255);
GetWindowText(this.Handle, sTitleBar, sTitleBar.Capacity);
MessageBox.Show(sTitleBar.ToString());
```

Таким образом, ответом на вопрос, как вызывать функцию, у которой есть выходной строковый параметр, будет — используйте класс `StringBuilder`.

# Изменение типа, применяемого для маршалинга по умолчанию

Например, мы хотим вызвать функцию `GetClassName`, которая принимает параметр `LPSTR` (`char*`) даже в Unicode-версиях. Если вы передадите строку, общеязыковая исполняющая среда (CLR) преобразует ее в серию `TCHAR`. Но с помощью атрибута `MarshalAs` можно переопределить то, что предлагается по умолчанию, как показано в листинге 17.4.

**Листинг 17.4. Переопределение типов при помощи `MarshalAs`**

```
[DllImport("user32.dll")]
public static extern int GetClassName(IntPtr hwnd,
    [MarshalAs(UnmanagedType.LPStr)] StringBuilder buf,
    int nMaxCount);
```

Теперь, когда вы вызовете `GetClassName`, .NET передаст вашу строку в виде символов ANSI, а не "широких символов".

# Вызов функций, требующих *struct*

Возьмем для примера функцию `GetWindowRect`, которая записывает в структуру `RECT` экранные координаты окна. Чтобы вызвать функцию `GetWindowRect` и передать ей структуру `RECT`, нужно использовать тип `struct` в сочетании с атрибутом `StructLayout`, как показано в листинге 17.5.

**Листинг 17.5. Использование типа `struct`**

```
[StructLayout(LayoutKind.Sequential)]
public struct RECT
{
    public int left ;
    public int top;
    public int right;
    public int bottom;
}

[DllImport("user32.dll")]
public static extern int GetWindowRect(IntPtr hwnd, ref RECT rc);
```

Важно использовать `ref`, чтобы CLR передала параметр типа `RECT` как ссылку. В этом случае функция сможет модифицировать ваш объект, а не его безымянную копию в стеке. После такого объявления функции можно ее вызвать в коде (листинг 17.6).

**Листинг 17.6. Использование структуры при вызове функции Windows API**

```
int w, h;
RECT rc = new RECT();
GetWindowRect(this.Handle, ref rc);
w = rc.right - rc.left;
h = rc.bottom - rc.top;
MessageBox.Show("Ширина формы: " + w + "\n\rВысота формы: " + h);
```

Обратите внимание, что `ref` используется и в объявлении, и при вызове функции. Тип, по умолчанию применяемый для маршалинга типов `struct` — `LPStruct`, поэтому необходимости в атрибуте `MarshalAs` нет. Но если вы хотите использовать `RECT` в виде класса, а не `struct`, вам необходимо реализовать оболочку, показанную в листинге 17.7.

**Листинг 17.7. Использование структуры в виде класса**

```
// Если RECT - класс, а не структура (struct)
[DllImport("user32.dll")]
public static extern int GetWindowRect(IntPtr hwnd,
[MarshalAs(UnmanagedType.LPStruct)] RECT rc);
```

# Работа с функциями обратного вызова в C#

Для использования функций, написанных на C#, в качестве функций обратного вызова Windows нужно использовать делегаты (`delegate`). Пример показан в листинге 17.8.

**Листинг 17.8. Использование делегатов для функции обратного вызова**

```
delegate bool EnumWindowsCB(int hwnd, int lparam);
```

Объявив свой тип делегата, можно написать оболочку для функции Windows API, как это сделано в листинге 17.9.

**Листинг 17.9. Оболочка для функции Windows API**

```
[DllImport("user32")]
public static extern int EnumWindows(EnumWindowsCB cb, int lparam);
```

# Создание собственной управляемой библиотеки

Можно создать собственную управляемую библиотеку, из которой можно будет вызывать функции Windows API. Для этого в Visual Studio предусмотрены специальные опции. Новый проект создается как библиотека классов (Class Library). Сборка при этом автоматически получает расширение dll. Использовать управляемую библиотеку в управляемом коде просто. Для этого надо добавить ссылку (используя меню **Project | Add Reference…**) на библиотечную сборку, указав месторасположение сборки в соответствующем диалоговом окне. После этого Visual Studio копирует сборку в директорию, в которой располагается разрабатываемый код. Далее в коде программы используется либо оператор `using`, либо полное имя библиотечного модуля с

точечной нотацией. Все библиотечные классы и методы готовы к использованию в коде программы.

***Примечание***

Пример создания библиотеки и использования функций Windows API находится в папке Win32 на прилагаемом диске.

# Примеры использования функций API

Вкратце ознакомившись с теорией, перейдем к конкретным примерам. В предыдущих главах я уже неоднократно приводил пример использования функций Windows API для решения различных проблем. Рассмотрим еще несколько полезных советов, которые не вошли в другие главы.

## Блокировка компьютера

Если вам необходимо блокировать компьютер, то вызовите функцию `LockWorkStation` (листинг 17.10). Результат работы будет аналогичен нажатию комбинации клавиш <Win>+<L> или <Ctrl>+<Alt>+<Del> с последующим выбором кнопки (или команды меню) **Блокировка**.

**Листинг 17.10. Блокировка компьютера**

```
// Функция для блокировки компьютера
[DllImport("user32.dll")]
private static extern void LockWorkStation();

// Блокируем компьютер
LockWorkStation();
```

## Является ли текущий пользователь администратором?

Если необходимо удостовериться, что текущий пользователь имеет права администратора, то можно вызвать функцию `IsUserAnAdmin`, как показано в листинге 17.11.

**Листинг 17.11. Проверка текущего пользователя**

```
[DllImport("shell32.dll")]
public static extern bool IsUserAnAdmin();

private void butIsAdmin_Click(object sender, EventArgs e)
{
    MessageBox.Show(IsUserAnAdmin().ToString());
}
```

## Мигание заголовка формы

Наверное, вам приходилось видеть, что заголовок окна вдруг начинал мигать, привлекая ваше внимание. Подобный эффект реализуется вызовом функций `FlashWindow` или `FlashWindowsEx`. Я решил показать вам пример использования функции `FlashWindowEx`, как более мощной и современной. Поместите на форму кнопку и напишите код, приведенный в листинге 17.12.

**Листинг 17.12. Мигание кнопки программы в панели задач**

```
// Функция, константы и структура для мигания окна
public const int FLASHW_STOP = 0;
public const int FLASHW_CAPTION = 0x00000001;
public const int FLASHW_TRAY = 0x00000002;
public const int FLASHW_ALL = (FLASHW_CAPTION | FLASHW_TRAY);
public const int FLASHW_TIMER = 0x00000004;
public const int FLASHW_TIMERNOFG = 0x0000000C;

[StructLayout(LayoutKind.Sequential)]
public struct FLASHWINFO
{
    [MarshalAs(UnmanagedType.U4)]
    public int cbSize;
    public IntPtr hwnd;
    [MarshalAs(UnmanagedType.U4)]
    public int dwFlags;
    [MarshalAs(UnmanagedType.U4)]
    public int uCount;
    [MarshalAs(UnmanagedType.U4)]
    public int dwTimeout;
}
```

```
[DllImport("user32.dll")]
public static extern bool FlashWindowEx([MarshalAs(UnmanagedType.Struct)]
ref FLASHWINFO pfwi);

private void butFlashWindow_Click(object sender, EventArgs e)
{
    FLASHWINFO fwi = new FLASHWINFO();

    fwi.cbSize = Marshal.SizeOf(fwi);
    fwi.hwnd = this.Handle;
    fwi.dwFlags = FLASHW_TRAY;
    fwi.dwTimeout = 0;
    fwi.uCount = 5;

    FlashWindowEx(ref fwi);
}
```

В нашем примере используется константа `FLASHW_TRAY`, что позволяет выбрать вариант мигания заголовка окна только на панели задач. Когда вы нажмете на соответствующую кнопку, то форма должна мигнуть 5 раз (переменная `uCount`). Интересно отметить, что в бета-версии Visual Studio 2005 была возможность использовать мигание встроенными средствами, но затем эту функциональность убрали, хотя и обещали вернуться к ней в следующих версиях Visual Studio.

## Форматирование дисков

Чтобы вызвать стандартное диалоговое окно форматирования дисков, нужно воспользоваться функцией `SHFormatDrive`. Пример приведен в листинге 17.13.

**Листинг 17.13. Вызов диалогового окна форматирования дисков**

```
[DllImport("shell32.dll")]
static extern uint SHFormatDrive(IntPtr hwnd,
uint drive,
uint fmtID,
uint options);
```

```
private void butFormatDiskDial_Click(object sender, EventArgs e)
{
    SHFormatDrive(this.Handle, 3, 0, 0);
}
```

## Открытие и закрытие лотка привода компакт-дисков

Наверное, при работе с утилитами, прожигающими компакт-диски CD-R и CD-RW, вы замечали, что у них имеется возможность извлекать компакт-диск из привода программным путем. Неплохо бы научиться делать то же самое при помощи C#. Для этого используем функцию `mciSendString` в связке со специальными командами, которые и позволят нам открывать и закрывать лоток привода компакт-дисков (листинг 17.14).

**Листинг 17.14. Открытие и закрытие лотка привода компакт-диска**

```
// Функция для открытия и закрытия лотка привода CD
[DllImport("winmm.dll", EntryPoint = "mciSendStringA",
CharSet = CharSet.Ansi)]
public static extern int mciSendString(string strCommand,
StringBuilder strReturnString,
int cchReturn, IntPtr hwndCallback);

private void butOpenCD_Click(object sender, EventArgs e)
{
    // Открываем лоток
    mciSendString("set CDAudio door open", null, 0, IntPtr.Zero);
}

private void butCloseCD_Click(object sender, EventArgs e)
{
    // Закрываем лоток
    mciSendString("set CDAudio door closed", null, 0, IntPtr.Zero);
}
```

***Примечание***

Примеры работы с функциями Windows API находятся в папке WinAPIDemo на прилагаемом компакт-диске.

## Создание собственного пункта в системном меню

У большинства окон в Windows имеется так называемое системное меню. Внешний вид, как правило, у всех меню одинаков, но иногда попадаются программы, у которых в системном меню имеются свои собственные пункты. Естественно, любого программиста разбирает любопытство — а как реализовать эту функциональность в своей программе. На данный момент .NET Framework не предоставляет такого полезного свойства, как `Form.SystemMenu` или что-то в этом роде. Поэтому придется прибегать к помощи механизма P/Invoke для вызовов функций Windows API. Опытные программисты (особенно имеющие опыт работы с языком C++) знают, что для модификации системного меню используется функция `GetSystemMenu`, а также вспомогательная функция `AppendMenu`, которая позволяет добавлять в меню разделители, картинки, галочки и сам текст. Итак, объявим необходимые функции с применением атрибута `DllImport`, а также другие переменные (листинг 17.15).

**Листинг 17.15. Объявление функций для системного меню**

```
// Флаги для системного меню
public enum MenuFlags
{
    MF_BITMAP = 0x00000004,
    MF_CHECKED = 0x00000008,
    MF_DISABLED = 0x00000002,
    MF_ENABLED = 0x00000000,
    MF_GRAYED = 0x00000001,
    MF_MENUBREAK = 0x00000020,
    MF_OWNERDRAW = 0x00000100,
    MF_POPUP = 0x00000010,
    MF_SEPARATOR = 0x00000800,
    MF_STRING = 0x00000000,
    MF_UNCHECKED = 0x00000000
}

[DllImport("user32.dll", EntryPoint = "GetSystemMenu",
SetLastError = true, CharSet = CharSet.Unicode, ExactSpelling = true,
CallingConvention = CallingConvention.Winapi)]
private static extern IntPtr GetSystemMenu(IntPtr hwnd,
int bRevert);
```

```
[DllImport("user32.dll")]
private static extern bool AppendMenu(IntPtr hMenu,
MenuFlags uFlags,
uint wIDNewItem, String lpNewItem);

// Сообщение Windows
const int WM_SYSCOMMAND = 0x0112;

// Наш новый идентификатор для системного меню
const int ID_ABOUT = 1000;
```

Чтобы усилить контроль типов, мы использовали перечисление `MenuFlags` для функции `AppendMenu`. Кроме того, применение перечисления позволяет использовать возможности IntellSense, что ускоряет написание кода. Для создаваемого нового пункта меню мы задаем новый идентификатор `ID_ABOUT`. Важно, чтобы значение идентификатора не вступило в конфликт с встроенными системными командами. Теперь у нас все готово к добавлению нового пункта в меню. Для этого в конструкторе формы пишем код, приведенный в листинге 17.16.

**Листинг 17.16. Создание нового пункта в системном меню**

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

    // Получаем описатель системного меню
    IntPtr hSysMenu = GetSystemMenu(this.Handle, 0);

    // Добавляем новые пункты в системное меню
    AppendMenu(hSysMenu, MenuFlags.MF_SEPARATOR, 0, null);
    AppendMenu(hSysMenu, MenuFlags.MF_STRING, ID_ABOUT,
               "C#.Народные советы");
}
```

Сначала мы получаем описатель системного меню, а затем добавляем нужную команду. Если вы запустите проект на этой стадии, то увидите, что в системном меню появилась новая команда. Но если вы щелкнете на этом пункте, то ничего не произойдет. Это нас не устраивает. Нужно переопределить виртуальный метод `WndProc` в нашем классе для обработки щелчка мыши по добавленному пункту меню, как это сделано в листинге 17.17.

**Листинг 17.17. Обработка щелчка мыши**

```
protected override void WndProc(ref Message msg)
{
    if (msg.Msg == WM_SYSCOMMAND)
    {
        if(msg.WParam.ToInt32()==ID_ABOUT)
        {
            MessageBox.Show("Вы выбрали пункт системного меню");
            return;
        }
    }
    base.WndProc(ref msg);
}
```

Запустите проект и проверьте работу приложения (рис. 17.1). Сейчас все работает, как положено. Теперь вы знаете, как вставлять новые пункты в системное меню. Для более профессионального кода можно также использовать функции `GetMenuItemCount` и `RemoveMenu`, с помощью которых можно подсчитать число команд в меню, удалить лишние команды и вставить свою команду в нужное место. Попробуйте реализовать эту задачу самостоятельно.

### *Примечание*

Пример добавления команды в системное меню находится в папке SystemMenu на прилагаемом диске.

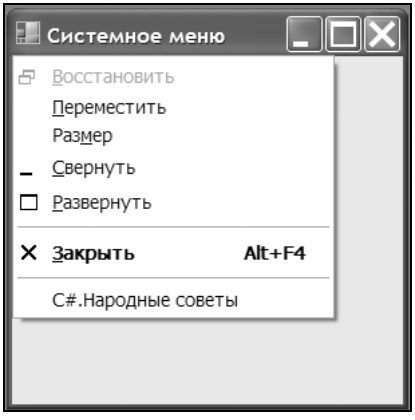



**Рис. 17.1.** Новая команда в системном меню

## Работа с конфигурационными файлами INI

Конфигурационные файлы используются для сохранения различных настроек программы, которыми можно воспользоваться при открытии нового сеанса работы с этой программой. Например, мы хотим запомнить размеры окна, его цвет, текст в текстовых полях и другие настройки. Можно придумать свой вариант хранения данных, разработав собственный формат файлов. А можно воспользоваться готовыми решениями. Конфигурационные файлы с расширением INI использовались еще в старых версиях Windows, например, в Windows 3.11. Затем специалисты Microsoft решили хранить настройки в реестре и предложили разработчикам отказаться от файлов INI. В настоящее время стало модным использовать файлы XML. У всех этих способов есть свои плюсы и минусы. К плюсам файлов INI можно отнести простоту редактирования файла, который является обычным текстовым файлом, возможность создания переносимых приложений, которые не засоряют реестр, а также их малый размер по сравнению с файлами XML. Рассмотрим пример работы с файлами INI при помощи функций Windows API `WritePrivateProfileString`, `GetPrivateProfileSting` и `GetPrivateProfileInt`. Предположим, что мы хотим запоминать размеры формы, установленные пользователем, а также текст, введенный пользователем в текстовый файл, чтобы при следующей загрузке программы использовать эти значения. Именно это и делается в примере, приведенном в листинге 17.18.

**Листинг 17.18. Работа с конфигурационными файлами**

```
[DllImport("kernel32")]
private static extern bool WritePrivateProfileString(string AppName,
string KeyName,
string lpString, string FileName);

[DllImport("kernel32")]
private static extern int GetPrivateProfileString(string AppName,
string KeyName, string lpDefault,
StringBuilder ReturnedString,
int Size, string FileName);

[DllImport("kernel32.dll")]
private static extern int GetPrivateProfileInt(string AppName,
string KeyName, int nDefault, string FileName);
```

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    // Устанавливаем значение ключей Width и Height в разделе [Form]
    // файла C:\config.ini в соответствии с размерами формы
    WritePrivateProfileString("Form",
        "Width", this.Width.ToString(), @"C:\config.ini");
    WritePrivateProfileString("Form",
        "Height", this.Height.ToString(), @"C:\config.ini");

    // Запоминаем текст в текстовом поле в разделе [Text] файла
    // C:\config.ini
    WritePrivateProfileString("TextBox1",
        "Text", textBox1.Text, @"C:\config.ini");
}

private void Form1_Load(object sender, EventArgs e)
{
    // читаем значение "Text" в секции [TextBox1]
    // INI-файла config.ini
    StringBuilder sb = new StringBuilder(256);
    GetPrivateProfileString("TextBox1", "Text",
        "C#.Народные советы", sb, sb.Capacity, @"C:\config.ini");

    textBox1.Text = sb.ToString();

    // Читаем значение Width из ключа [Form]
    // файла INI file C:\CONFIG.INI
    this.Width =
        GetPrivateProfileInt("Form", "Width", 200, @"C:\config.ini");

    // Читаем значение Height из ключа [Form]
    // файла INI file C:\CONFIG.INI
    this.Height =
        GetPrivateProfileInt("Form", "Height", 200, @"C:\config.ini");
}
```

Запустите пример и поиграйте с размерами формы и текстом в текстовом поле. Когда убедитесь, что все работает, проделайте следующий трюк. Удалите созданный конфигурационный файл c:\config.ini, из которого считывает нужные данные программа, запустите снова программу и посмотрите на результат. Вы увидите, что форма имеет размеры, не соответствующие размерам в среде раз-

работки, а также некоторый текст в текстовом поле. Как вы уже догадались, если программа не может обнаружить нужный конфигурационный файл, то она использует предопределенные значения, которые указаны в третьем параметре функций `GetPrivateProfileString` и `GetPrivateProfileInt`.

> ***Примечание***
>
> Пример работы с INI-файлами находится в папке INIFiles на прилагаемом диске.

## Извлечение значков из файлов

Функция `ExtractIcon` позволяет извлекать значки из ресурсов, которые зашиты в файлах EXE, DLL, CPL и др. Кроме того, функция позволяет подсчитать количество значков, находящихся в файле. В качестве испытуемого файла возьмем динамическую библиотеку shell32.dll, которая имеется в любой версии Windows. Для удобства в примере (листинг 17.19) я создал две отдельные процедуры `GetNumberOfIcons` и `ExtractIconsFromFile`. Первая процедура позволяет узнать число значков, содержащихся в файле, а вторая может извлечь нужный значок и показать ее на форме.

**Листинг 17.19. Извлечение значка из NOTEPAD.EXE**

```
[DllImport("shell32.dll")]
static extern IntPtr ExtractIcon(IntPtr hInst,
    string sFileName, int nIconIndex);

string filename = @"C:\windows\system32\shell32.dll";

private int GetNumberOfIcon(string str)
{
    return (int)ExtractIcon(this.Handle, str, -1);
}

/// <summary>
/// Извлекает значок из указанного файла в заданную позицию.
/// </summary>
/// <param name="str">Путь к файлу со значками.</param>
/// <param name="index">Номер значка в файле.</param>
/// <param name="x">Координата X.</param>
/// <param name="y">Координата Y.</param>
```

```
private void ExtractIconsFromFile(string str, int index, int x, int y)
{
    IntPtr retval;

    retval = ExtractIcon(this.Handle, str, index);

    Icon icon = Icon.FromHandle(retval);
    Graphics g = CreateGraphics();
    g.DrawIcon(icon, x, y);
    g.Dispose();
}

private void Form1_Load(object sender, EventArgs e)
{
    this.Text = GetNumberOfIcon(filename).ToString();
}

private void Form1_Paint(object sender, PaintEventArgs e)
{
    // Выводим шестнадцатый значок из файла shell32.dll
    ExtractIconsFromFile(filename, 15, 45, 25);
}
```

Чтобы подсчитать число значков в файле, достаточно указать процедуре `GetNumberOfIcons` полный путь к файлу. Как видите, в файле shell32.dll содержится более 200 значков. Если вы хотите посмотреть на какой-нибудь из этих значков, то извлекайте его из файла и рисуйте на форме с помощью метода `ExtractIconsFromFile`. В нашем примере я вывожу на форму в позицию (45, 25) шестнадцатый значок. Таким образом, вы можете заглянуть вовнутрь любых файлов со значками, просмотреть их и при необходимости даже использовать их в своих программах. Например, понравившийся вам значок можно вывести в качестве своего значка на форму.

## Вызов диалогового окна *Смена значка*

Продолжим работать со значками. Существует такая функция Windows API, как `PickIconDlg`. Долгое время она была официально не документирована, но, начиная с Windows 2000, компания Microsoft все-таки выложила описание этой функции на сайте MSDN. Функция `PickIconDlg` вызывает стандартное диалоговое окно **Смена значка**, позволяющее выбрать значок из модуля.

Тем самым мы можем предоставить пользователю возможность самому выбирать нужный значок, после чего вывести его на форму (или произвести другие операции). В листинге 17.20 показано, как это может выглядеть в программе.

**Листинг 17.20. Вызов диалогового окна Смена значка**

```
[DllImport("shell32.dll", CharSet = CharSet.Auto, CallingConvention =
CallingConvention.Winapi)]
private static extern int PickIconDlg(IntPtr hwndOwner,
string FileName, int nMaxFile, ref int lpdwIconIndex);

private void butShowPickDial_Click(object sender, EventArgs e)
{
    // Выводим диалоговое окно Смена значка
    // и показываем выбранный значок в PictureBox

    // Индекс выбираемого значка при открытии диалогового окна
    int iconindex = 3;

    // Выводим диалоговое окно
    PickIconDlg(this.Handle, filename, 0, ref iconindex);

    IntPtr retval;
    // Извлекаем значок по индексу
    retval = ExtractIcon(this.Handle, filename, iconindex);

    // Выводим в заголовке номер индекса (для проверки)
    this.Text = iconindex.ToString();

    // Выводим значок в рисуночной области
    Icon icon = Icon.FromHandle(retval);
    Graphics g = pictureBox1.CreateGraphics();
    g.DrawIconUnstretched(icon, new Rectangle(0, 0, 50, 50));
    g.Dispose();
}
```

## Продолжаем работать со значками

Вы еще не устали? Тогда я расскажу вам еще об одной функции Windows API, а именно о функции `SHGetFileInfo`, с помощью которой также можно

работать со значками, хотя возможности функции этим не ограничиваются. Ее применение для извлечения значка из приложения mspaint.exe показано в листинге 17.21.

**Листинг 17.21. Вызов функции `SHGetFileInfo`**

```
[DllImport("shell32.dll")]
public static extern IntPtr SHGetFileInfo(string pszPath,
uint dwFileAttributes,
ref SHFILEINFO psfi,
uint cbSizeFileInfo,
uint uFlags);

public struct SHFILEINFO
{
    public IntPtr hIcon;
    public IntPtr iIcon;
    public uint dwAttributes;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 260)]
    public string szDisplayName;
    [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 80)]
    public string szTypeName;
};

public const uint SHGFI_ICON = 0x100;
public const uint SHGFI_LARGEICON = 0x0;
public const uint SHGFI_SMALLICON = 0x1;

private void butSHGetFileInfo_Click(object sender, EventArgs e)
{
    int nIndex = 0;
    IntPtr hImgSmall;
    string fName = @"c:\windows\system32\mspaint.exe";
    SHFILEINFO shinfo = new SHFILEINFO();

    listView1.SmallImageList = imageList1;
    listView1.LargeImageList = imageList1;

    // Получаем маленькие значки
    hImgSmall = SHGetFileInfo(fName, 0,
        ref shinfo, (uint)Marshal.SizeOf(shinfo),
        SHGFI_ICON | SHGFI_SMALLICON);
```

```
    //Полученный значок возвращается как член структуры shinfo
    Icon myIcon = Icon.FromHandle(shinfo.hIcon);

    imageList1.Images.Add(myIcon);

    //добавляем имя файла и значок в listview
    listView1.Items.Add(fName, nIndex++);
}
```

***ПРИМЕЧАНИЕ***

Примеры работы со значками находится в папке Icons на прилагаемом диске.

## Панель задач, кнопка *Пуск* и часы в области уведомлений

Очень часто программисты хотят получить доступ к стандартным элементам интерфейса Рабочего стола Windows. Например, разработчики хотят получить координаты панели задач, программно нажать на кнопку **Пуск**, спрятать и показать эту кнопку Пуск и многое другое. В примере, приведенном в листинге 17.22, я покажу, как это делается.

**Листинг 17.22. Работа с элементами Рабочего стола**

```
using System.Runtime.InteropServices;

[DllImport("user32.dll")]
private static extern IntPtr FindWindow(
string className,
string windowName);

[DllImport("user32.dll")]
private static extern IntPtr FindWindowEx(
IntPtr hwndParent,
IntPtr hwndChildAfter,
string className,
string windowName);

[DllImport("user32.dll")]
private static extern bool GetWindowRect(
```

```
IntPtr hWnd,
ref RECT rect);

[DllImport("shell32.dll")]
private static extern int SHAppBarMessage(
int flag,
ref AppBarData abd);

[DllImport("user32.dll")]
static extern bool ShowWindow(
IntPtr hWnd,
int nCmdShow);

[DllImport("user32.dll")]
public static extern IntPtr SendMessage(IntPtr hWnd,
UInt32 Msg, Int32 wParam, Int32 lParam);

const int WM_SYSCOMMAND = 0x0112;

private const int SC_TASKLIST = 0xF130;
private const int ABM_GETSTATE = 4;
private const int ABM_GETTASKBARPOS = 5;
private const int ABM_SETSTATE = 10;
private const int SW_HIDE = 0;
private const int SW_SHOW = 5;

// состояние Панели задач
private const int ABS_AUTOHIDE = 1;
private const int ABS_ALWAYSONTOP = 2;
private const int ABS_BOTH = 3;


[StructLayout(LayoutKind.Sequential)]
public struct AppBarData
{
    public int cbSize;
    public IntPtr hWnd;
    public int uCallbackMessage;
    public int uEdge;
    public RECT rc;
    public IntPtr lParam;
}
```

```
[StructLayout(LayoutKind.Sequential)]
public struct RECT
{
    public int left;
    public int top;
    public int right;
    public int bottom;
}

// Описатель Панели задач
IntPtr hW;

// Описатель кнопки Пуск
IntPtr hWndStart;

// Описатель области уведомлений
IntPtr tray;

private void butHideStart_Click(object sender, EventArgs e)
{
    // Ищем окно с классом Shell_TrayWnd
    hW = FindWindowEx(IntPtr.Zero, IntPtr.Zero, "Shell_TrayWnd",null);

    // Получим описатель кнопки Пуск
    hWndStart = FindWindowEx(hW, IntPtr.Zero, "BUTTON", null);
    // Прячем кнопку Пуск
    ShowWindow(hWndStart, SW_HIDE);
}

private void butShowStart_Click(object sender, EventArgs e)
{
    // Показываем кнопку Пуск
    ShowWindow(hWndStart, SW_SHOW);
}

private void butPressStart_Click(object sender, EventArgs e)
{
    // Нажимаем на кнопку Пуск
    SendMessage(this.Handle, WM_SYSCOMMAND, SC_TASKLIST, 0);
}
```

```
private void butHideTray_Click(object sender, EventArgs e)
{
    hW = FindWindowEx(IntPtr.Zero, IntPtr.Zero, "Shell_TrayWnd", null);

    // Дескриптор области уведомлений
    tray = FindWindowEx(hW, IntPtr.Zero, "TrayNotifyWnd", null);

    // Прячем область уведомлений
    ShowWindow(tray, SW_HIDE);
}

private void butShowTray_Click(object sender, EventArgs e)
{
    ShowWindow(tray, SW_SHOW);
}

private void butHideClock_Click(object sender, EventArgs e)
{
    // Описатель панели задач
    hW = FindWindowEx(IntPtr.Zero, IntPtr.Zero, "Shell_TrayWnd", null);

    // Описатель области уведомлений
    tray = FindWindowEx(hW, IntPtr.Zero, "TrayNotifyWnd", null);

    // Описатель системных часов
    IntPtr trayclock = FindWindowEx(tray, IntPtr.Zero,
                                    "TrayClockWClass", null);

    // Прячем системные часы
    ShowWindow(trayclock, SW_HIDE);
}

private void butTaskbar_Click(object sender, EventArgs e)
{
    AppBarData bardata = new AppBarData();
    bardata.cbSize = Marshal.SizeOf(typeof(AppBarData));
    bardata.hWnd = Handle;

    int retval = SHAppBarMessage(ABM_GETSTATE, ref bardata);

    switch (retval)
    {
```

```
        case ABS_AUTOHIDE:
            MessageBox.Show("Автоматически убирать с экрана");
            break;
        case ABS_ALWAYSONTOP:
            MessageBox.Show("Всегда на экране");
            break;
        case ABS_BOTH:
            MessageBox.Show("Включена опция Автоматически убирать " +
                "с экрана и Показывать всегда поверх окон");
            break;
    }

    // Получим позицию Панели задач
    SHAppBarMessage (ABM_GETTASKBARPOS, ref bardata);

    // Получим координаты ограничивающего прямоугольника
    MessageBox.Show("Координаты Панели задач: (" +
        bardata.rc.left + "," + bardata.rc.top + ")" +
        "-" + "(" +
        bardata.rc.right + "," + bardata.rc.bottom + ")");
}
```

***ПРИМЕЧАНИЕ***

Пример работы с элементами Рабочего стола находится в папке TaskBarAPI на прилагаемом диске.

## Смена обоев Рабочего стола

Если вы хотите периодически менять картинку на своем Рабочем столе, то можете это сделать программным способом прямо из своего приложения. Для смены обоев Рабочего стола вызывается одна функция Windows API `SystemParametersInfo`. Общий принцип смены обоев не сложен. Во-первых, вам нужно знать путь к изображению, которое будет служить новыми обоями Рабочего стола. Выбранная картинка должна быть в формате BMP, поэтому вам может понадобиться сохранить любимую фотографию в формате JPG в нужном формате при помощи метода `Save` класса `Bitmap`. Во-вторых, нужно внести некоторые изменения в реестр. Картинку можно использовать в качестве обоев тремя способами: по центру экрана, растянуть на весь экран или замостить Рабочий стол. После этих приготовлений можно вызывать функцию `SystemParametersInfo`, которая установит новую картинку в качестве

обоев Рабочего стола. Для упрощения примера (листинг 17.23) я жестко прописал путь к картинке.

**Листинг 17.23. Установка обоев Рабочего стола**

```
using System.Runtime.InteropServices;
using Microsoft.Win32;

[DllImport("user32.dll", CharSet = CharSet.Auto)]
static extern int SystemParametersInfo(
    int uAction, int uParam, string lpvParam, int fuWinIni);

// Константы
const int SPI_SETDESKWALLPAPER = 20;
const int SPIF_UPDATEINIFILE = 0x01;
const int SPIF_SENDWININICHANGE = 0x02;

public enum WallpaperStyle : int
{
    Tiled, Centered, Stretched
}

private void Form1_Load(object sender, EventArgs e)
{
    // связываем комбинированное окно с перечислением WallpaperStyle
    cboStyleWallpaper.DataSource =
        System.Enum.GetNames(typeof(WallpaperStyle));
}

private void butSetWallpaper_Click(object sender, EventArgs e)
{
    string imageFileName;
    imageFileName = Application.StartupPath + "/mycat.bmp";

    RegistryKey key = Registry.CurrentUser.OpenSubKey(
        "Control Panel\\Desktop", true);

    WallpaperStyle style;

    style = (WallpaperStyle)Enum.Parse(typeof(WallpaperStyle),
        cboStyleWallpaper.Text);
```

```
    switch (style)
    {
        case WallpaperStyle.Stretched:
            key.SetValue(@"WallpaperStyle", "2");
            key.SetValue(@"TileWallpaper", "0");
            break;

        case WallpaperStyle.Centered:
            key.SetValue(@"WallpaperStyle", "1");
            key.SetValue(@"TileWallpaper", "0");
            break;

        case WallpaperStyle.Tiled:
            key.SetValue(@"WallpaperStyle", "1");
            key.SetValue(@"TileWallpaper", "1");
            break;
    }

    SystemParametersInfo(SPI_SETDESKWALLPAPER, 0,
        imageFileName, SPIF_UPDATEINIFILE | SPIF_SENDWININICHANGE);
}
```


**Рис. 17.2.** Установка новых обоев на Рабочем столе

Сверните все имеющиеся окна на вашем Рабочем столе и запустите проект. Нажмите на кнопку, и вы сразу увидите результат. Ваш Рабочий стол преобразится (рис. 17.2).

> ***Примечание***
>
> Пример смены обоев Рабочего стола находится в папке Wallpaper на прилагаемом диске.

# Использование функций обратного вызова

В *разд. " Работа с функциями обратного вызова в C#"* мы рассказали, как можно использовать функции обратного вызова, написанные на C#. Приведем теперь один пример.

## Получение списка кодовых страниц, установленных в системе

Для перечисления установленных кодовых страниц можно использовать функцию `EnumSystemCodePages`, как показано в листинге 17.24.

**Листинг 17.24. Получение списка кодовых страниц, установленных в системе**

```
[DllImport("Kernel32.dll")]
static extern bool EnumSystemCodePages(CODEPAGE_ENUMPROC
lpLocaleEnumProc, int dwFlags);
delegate bool CODEPAGE_ENUMPROC(string lpLocaleString);

const int CP_INSTALLED = 0x00000001;

private bool EnumCodePagesProc(string locale)
{
    listBox1.Items.Add(locale);
    return true;
}
```

```
private void button1_Click(object sender, EventArgs e)
{
    EnumSystemCodePages(new CODEPAGE_ENUMPROC(EnumCodePagesProc),
        CP_INSTALLED);
}
```

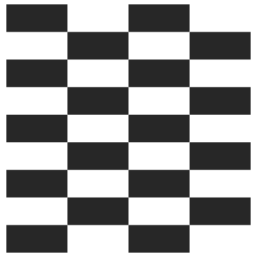
***ПРИМЕЧАНИЕ***

Пример получения списка кодовых страниц находится в папке CodePages на прилагаемом диске.

# Заключение

Несмотря на огромное число имеющихся классов .NET Framework, программисту по-прежнему приходится прибегать к вызовам системных функций Windows API. В папке Win32Help на прилагаемом к книге компакт-диске вы найдете демо-версию справочника по функциям Windows API для .NET Framework. Если вам понравится этот справочник, то вы можете приобрести его полную версию на моем сайте **http://netsources.narod.ru**.

# Глава 18



# Новинки Visual Studio 2008

Microsoft продолжает развивать свою платформу .NET Framework. Кажется, совсем недавно была выпущена для ознакомления бета-версия .NET Framework 1.0, а уже на подходе новая версия .NET Framework 3.5. Значит, снова появятся новые советы, трюки и решения. Самое главное нововведение, которое у всех на слуху, — это новый язык запросов LINQ, которым уже сейчас восторгаются многие программисты в своих блогах. Также рассмотрим и другие новшества, которые пришли к нам с выходом пакета разработки Visual Studio 2008.

## Новшества в C# 3.0

### Неявно типизированные переменные

В C# 3.0 можно объявить переменную без объявления ее типа при помощи ключевого слова `var`. Компилятор самостоятельно определит тип переменной, исходя из представленных данных. Например, можно написать следующий код:

```
var x = 5;
```

Компилятор должен догадаться, что переменная `x` имеет тип `int`, то есть наш код равносилен старому коду:

```
int x = 5;
```

Обязательно нужно помнить, что переменную надо сразу инициализировать, так как такой код работать не будет:

```
var y;
y = 7;
```

Аналогичным образом можно объявлять неявно типизированные массивы, что проиллюстрировано в листинге 18.1.

**Листинг 18.1. Объявление неявно типизированных массивов**

```
var arrInt = new[]{1, 2,,3}; // вместо int[] arrInt = ...
var arrDouble = new[]{1, 2, 3.0};  // вместо double arrDouble[] = ...
```

Тип массива в этих случаях определяется по типам элементов.

## Инициализация объектов

С помощью ключевого слова `var` также можно инициализировать объекты так, как показано в листинге 18.2.

**Листинг 18.2. Инициализация объектов**

```
var p = new Programer()
{
    p.FirstName = "Alexander";
    p.LastName = "Klimoff";
}

// Этот код равносилен следующему
Programer p = new Programer();
p.FirstName = "Alexander";
p.LastName = "Klimoff";
```

Это особенно удобно, если имя класса слишком длинное. Например, вместо старого способа объявления переменной можно применить короткий вариант (листинг 18.3).

**Листинг 18.3. Короткий вариант объявления переменной**

```
// Старый вариант объявления переменной
VeryLongFooClassWithTemplate<MyLongTypeParameter> local =
    new VeryLongFooClassWithTemplate<MyLongTypeParameter>();

// Новый краткий вариант
var localVar = new MyLongFooClassWithTemplate<MyLongTypeParameter>();
```

Таким образом, новое ключевое слово `var` вместо типа переменной говорит компилятору о том, что тип выражения надо вычислить из правой части. Не-

обходимо помнить, что с ключевым словом `var` работает именно компилятор, то есть определение типа работает не во время выполнения кода, а на этапе компиляции. Поэтому все ошибки, связанные с неправильным использованием типов, будут обнаружены на этапе компиляции.

# LINQ

LINQ (Language Integrated Query, запрос, интегрированный в язык программирования) представляет собой новую технологию обработки данных на уровне языка программирования. Главная особенность — синтаксис языка стал очень похож на язык запросов SQL. Нас интересует реализация этой технологии на языке C#. Рассмотрим несколько базовых примеров, которые дадут нам представление о мощи и удобстве использования технологии LINQ. Но более подробно о технологии LINQ вам придется прочесть в других книжках.

## Вывод чисел из заданного массива с условием

В примере, приведенном в листинге 18.4, мы выводим на экран целые числа из заданного массива, значения которых меньше или равны 8.

**Листинг 18.4. Вывод чисел из заданного массива с условием**

```
// Задаем массив чисел
int[] numbers = { 12, 14, 21, 3, 9, 8, 6, 7, 2, 10, 1, 2, 3, 4 };

// Выбираем из них числа, меньше чем 8
var lowNums =
    from n in numbers
    where n <= 8
    select n;

foreach (var x in lowNums)
{
    // Выводим числа, подходящие нашему условию
    textBox1.Text += x.ToString() + " ";
}
```

## Ключевое слово *Where*

В этом примере (листинг 18.5) используется ключевое слово `Where` для печати имени каждого числа от 0 до 9, у которого длина имени числа меньше, чем его значение. В этом случае код, переданный как лямбда-выражение, конвертируется в нужный тип.

**Листинг 18.5. Использование ключевого слова `Where`**

```
public void Linq2()
{
    string[] digits = { "ноль", "один", "два", "три", "четыре",
                        "пять", "шесть", "семь", "восемь", "девять" };

    var shortDigits =
        digits.Where((digit, index) => digit.Length < index);


    foreach (var d in shortDigits)
    {
        textBox1.Text +=
            String.Format("Длина слова {0} меньше его значения.", d) +
            Environment.NewLine;
    }
}
```

## Увеличение на единицу ряда чисел

В листинге 18.6 мы выводим на экран ряд чисел, бóльших на единицу, чем ряд чисел, заданных в массиве. Для этого мы используем ключевое слово `select` для добавления единицы к каждому элементу массива.

**Листинг 18.6. Увеличение на единицу ряда чисел**

```
public void Linq3()
{
    int[] numbers = { 1, 3, 5, 7, 9, 11, 13, 15 };

    var numsPlusOne =
        from n in numbers
        select n + 1;
```

```
    foreach (var i in numsPlusOne)
    {
        textBox1.Text += i + " ";
    }
}
```

## Вывод имени числа

В этом примере (листинг 18.7) мы выводим имя каждого числа массива, индексированного во втором массиве, который содержит имена.

**Листинг 18.7. Вывод имени числа**

```
public void Linq4()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    string[] strings = { "ноль", "один", "два", "три", "четыре",
                         "пять", "шесть", "семь", "восемь", "девять" };

    var textNums =
        from n in numbers
        select strings[n];

    foreach (var s in textNums)
    {
        textBox1.Text += s + Environment.NewLine;
    }
}
```

В результате выполнения примера в поле `textBox1` будут добавлены слова "пять", "четыре", "один", "три", "девять", "восемь" и т. д., каждое в новой строке.

## Вывод строк из массива в разных регистрах

В этом примере (листинг 18.8) мы печатаем строки из массива в верхнем и нижнем регистре. Данный пример демонстрирует использование анонимных

типов в выбранном выражении. Анонимный тип имеет два поля — `Upper` и `Lower`, которые позволяют выводить строки в разных регистрах.

**Листинг 18.8. Вывод строк в разных регистрах**

```
public void Linq5()
{
    string[] words = { "МарТыШка", "оСеЛ", "КОЗел", "мИшка" };

    var upperLowerWords =
        from w in words
        select new { Upper = w.ToUpper(), Lower = w.ToLower() };

    foreach (var ul in upperLowerWords)
    {
        textBox1.Text +=
            String.Format("Верхний регистр: {0}, Нижний регистр: {1}",
                ul.Upper, ul.Lower) +
            Environment.NewLine;
    }
}
```

## Оператор *Take*

Оператор `Take` возвращает заданное количество элементов массива, начиная с первого (листинг 18.9).

**Листинг 18.9. Использование оператора `Take`**

```
public void Linq6()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
    // Берем первые три элемента массива
    var first3Numbers = numbers.Take(3);

    foreach (var n in first3Numbers)
    {
        textBox1.Text += n + Environment.NewLine;
    }
}
```

## Оператор *TakeWhile*

Оператор `TakeWhile` выполняет почти то же самое, что и `Take`, однако выбор заканчивается не после фиксированного числа элементов, а когда перестает выполняться определенное условие. В примере (листинг 18.10) мы проходим через элементы массива, пока число меньше шести.

**Листинг 18.10. Использование оператора `TakeWhile`**

```
public void Linq7()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var firstNumbersLessThan6 = numbers.TakeWhile(n => n < 6);

    MessageBox.Show("Первые числа, которые меньше 6:");
    foreach (var n in firstNumbersLessThan6)
    {
        textBox1.Text += n + Environment.NewLine;
    }
}
```

## Оператор *Skip*

Оператор `Skip`, наоборот, выдает все элементы, пропуская определенное количество начальных элементов массива. Например, в листинге 18.11 мы получаем все элементы массива кроме первых четырех элементов.

**Листинг 18.11. Использование оператора `Skip`**

```
public void Linq8()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var notFirst4Numbers = numbers.Skip(4);

    // Пропущены первые 4 элемента
    foreach (var n in notFirst4Numbers)
    {
        textBox1.Text += n + Environment.NewLine;
    }
}
```

## Оператор *SkipWhile*

Оператор `SkipWhile` выбирает все элементы массива, начиная с того, на котором условие перестало выполняться. Пример, показанный в листинге 18.12, печатает все элементы массива, пропуская элементы до тех пор, пока не попадется число, которое делится на 3 без остатка.

**Листинг 18.12. Использование оператора `SkipWhile`**

```
public void Linq9()
{
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var allButFirst3Numbers = numbers.SkipWhile(n => n % 3 != 0);

    textBox1.Text =
        "Последовательность элементов начинается с первого элемента, "
        + "которое делится без остатка на 3:"
        + Environment.NewLine;

    foreach (var n in allButFirst3Numbers)
    {
        textBox1.Text += n + Environment.NewLine;
    }
}
```

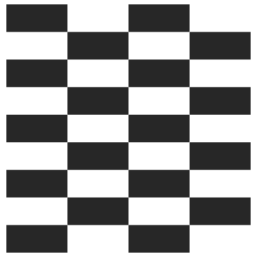
***ПРИМЕЧАНИЕ***

Пример работы с LINQ находятся в папке LINQ на прилагаемом диске.

# Заключение

Окончательная версия Visual Studio 2008 вышла в тот момент, когда работа над книгой подходила к концу. Из-за этого многие новые возможности не нашли описания в этой книге. Поэтому, уважаемый читатель, я вместе с вами буду изучать новинки, которые появились в новой версии .NET Frawework 3.5 и Visual Studio 2008.

Глава 19



# Ссылки на интересные места

Изучение любого языка состоит не только из чтения литературы и документации. Также необходимо просматривать информацию на различных сайтах, где опытные программисты делятся своим богатым опытом. В этой, последней главе книги я расскажу о тех сайтах и блогах, которые мне показались интересными и полезными для расширения своего кругозора.

## Сайты

Наверняка, многие из этих сайтов вам уже хорошо знакомы. Можете сравнить список посещаемых вами сайтов с моим.

- **http://www.codeproject.com** — англоязычный сайт CodeProject является, пожалуй, самым популярным сайтом среди программистов и пользуется уважением за качество представленных материалов. Здесь можно найти и сложный, профессионально написанный проект, и опыты начинающих новичков. Исходные коды программ выкладываются вместе со статьей, в которой автор подробно описывает особенности программы и при необходимости заостряет внимание на каких-то важных деталях. Часто проект сопровождается скриншотом программы, что тоже немаловажно для посетителей сайта. Иногда на страницах сайта встречаются и работы наших соотечественников, которые хорошо владеют английским языком и могут рассказать о своей разработке всему миру.
- **http://www.gotdotnet.ru** — этот сайт является одним из самых известных русскоязычных ресурсов в Рунете для программистов, использующих платформу .NET Framework. Особое внимание обратите на форумы сайта. На форумах вы можете узнать очень много полезной информации, а так-

же задать свои вопросы, которые весьма оперативно обрабатываются аудиторией сайта. Опытные посетители сайта стараются ответить на любые сложные вопросы и готовы поддерживать поднятую тему до бесконечности. Кстати, этот сайт был нашим ответом на англоязычный сайт **http://www.gotdotnet.com**, который пользовался популярностью у зарубежных программистов. К сожалению, на данный момент сайт прекратил свою работу, о чем можно прочитать на главной странице сайта.

- **http://www.rsdn.ru** — это еще один популярный российский сайт, на котором можно найти материалы по самой разной тематике. Имеется большое количество форумов на разные темы. Также издается журнал "RSDN Magazine", в котором печатаются интересные статьи.
- **http://snippets.dzone.com/tag/csharp** — как видно из названия, на этом англоязычном сайте собираются короткие полезные фрагменты кода, которые можно использовать в своих программах.
- **http://msdn.microsoft.com —** сайт Microsoft, который должен посещать любой пишущий под Windows программист для поиска самой свежей информации.

# Блоги

В последнее время я заметил, что интересные вещи все чаще можно узнать не на тематических сайтах, а на блогах, где авторы делятся своими мыслями, находками, кусочками кода. Читая рассуждения авторов интернет-дневников, порой натыкаешься на очень любопытные вещи. Причем, читая блог одного автора, можно перейти по ссылке на блог другого, и т. д. И процесс чтения интересных новостей затягивается до бесконечности. Я приведу лишь несколько адресов блогов, которые были упомянуты в этой книге.

- **http://seregaborzov.wordpress.com** — сетевой журнал Сергея Борзова, который делится своими собственными интересными примерами или дает ссылки на другие сайты.
- **http://andybor.blogspot.com** — блог Андрея Бороздина, который также делится полезной информацией с читателями.
- **http://blogs.gotdotnet.ru/personal/tk/default.aspx** — TK'S Blog. К сожалению, я не нашел информацию об авторе этого сетевого журнала. Но некоторые его заметки почитал с удовольствием.

- **http://dirtydogstink.com/blog/default.aspx** — блог Тода Хилтона (Tod Hilton), который является сотрудником компании Microsoft. Посетив его сетевой журнал, вы найдете несколько интересных статей о различных приемах программирования на C#.

# Заключение

Список любимых ресурсов можно продолжать до бесконечности. Но каждый должен сам собрать собственную коллекцию ссылок на сайты, которые ему интересны. В заключение, приведу еще одну ссылку — **http://netsources.narod.ru**. По этому адресу вы найдете мою страничку, на которой я буду выкладывать свежую информацию обо всех новинках C#. Добро пожаловать!

# Описание компакт-диска

Прилагаемый к книге компакт-диск содержит файлы проектов, которые находятся в папках в соответствии с главами книги. В таблице приводится краткое описание примеров и их расположения на диске.

| Глава | Описание | Папки с примерами |
|---|---|---|
| 2 | Примеры, связанные с числами, датами, строками | CompactPathDemo, DateSamples, EnumDemo, NumbersSampes, SimpleStrings, StringFX |
| 3 | Алгоритмы | Algorithm, Quine |
| 4 | Расширение для IDE | GhostDoc |
| 5 | Работа с формой | AboutScreen, ChangeResolution, CommandLine, DiamondForm, DragForm, DragFormAPI, DragWndProc, EasterEgg, FormsDemo, HoleForm, MDIBackColor, NoClose, NoForm, PassValue_1, PassValue_2, RemoveX, SingleInstance, TextForm, TransparentDemo, VisualStyleDemo |
| 6 | Использование различных элементов управления | ButtonDemo, ComboBoxDemo, Controls, ControlsDemo, DateTimePickerDemo, LabelAndPanel, LabelLine, LinkLabelDemo, ListBoxDemo, ListViewDemo, MeasureStringDemo, MenuDemo, NotifyIconDemo, OvalButton, PerformanceCounter, RichTextDemo, SmoothProgressBar, TabControl, TextBoxDemo, ToolTipDemo, TreeViewDemo |
| 7 | Работа с графикой | 3DText, AboutColor, Animate, CaptureScreen, ClipBoard, DrawRoundedRectangleDemo, FromResource, GetScreenColorDemo, ImageFX, InstalledFonts, InstalledPrinters, RotatedText, ScrollText |
| 8 | Примеры работы с мышью и клавиатурой | AdvancedMouse, AnimatedCursors, APIClicks, CapsLock, Keyboard, LastInput, MouseClick, MovingMouse, SendKeysDemo, SimpleMouse, SimplePaint, TwoButtons |

*(окончание)*

| Глава | Описание | Папки с примерами |
|---|---|---|
| 9 | Взаимодействие с другими приложениями | Application, ColorConsole, EventLogDemo, FileVersionInfoDemo, FrameworkFolder, NETFrameworkVersions, OperatingSystemDemo, ProcessesSamples, QueryPerformanceDemo, RegisterDLL, RegistryDemo, Settings, SystemEventsDemo, SystemInformationDemo |
| 10 | Работа с папками и файлами | AccessFile, DirectoryDemo, Encoding, FileCompare, FilesDemo, LogicalDisk, TreeViewDirectory, Watcher, XMLTextReaderDemo |
| 11 | Примеры использования WSH | WSHDemo |
| 12 | Примеры взаимодействия с WMI | GetHardwareInfo, WMIDemo |
| 13 | Работа с мультимедиа | BeepDemo, MP3TagEditor, MusicPlayer, SystemMedia |
| 14 | Глобализация и локализация | CultureExplorer, CultureInfo, Locale |
| 15 | Примеры работы с MS Office | EarlyBinding, ExcelDemo, LateBinding, OutlookDemo, OutlookNewMailEx |
| 16 | Использование локальной сети и сети Интернет | CheckUrlDemo, CurrentUser, DNS, FirefoxSample, FTPDemo, ListComputersLAN, NETPinger, NetRemoteTODSample, NetworkDemo, PingDemo, RequestResponse, SMTPMail, SQLEnum, URI, WinInet |
| 17 | Особенности работы с функциями Windows API | CodePages, Icons, INIFiles, SystemMenu, TaskBarAPI, Wallpaper, Win32, WinAPIDemo, Win32Help |
| 18 | Примеры использования LINQ | LINQ |
| Win32 Help | Демо-версия справочника по функциям Windows API для .NET Framework. Полную версию этого справочника можно приобрести на сайте **http://netsources.narod.ru** | |

# Предметный указатель

## A



## B



## C

## D



## E



## F

# G



# H



# I

# J



# K



# L

## M



## N



## O



## P

## Q



## R



## S

# T



# U



# V



# W

## X

# А



# Б



# В



# Г



# Д



# Ж

# З



# И



# К

## Л



## М

## Н



## О



## П

# P



# C

## T



## У

## Ф

## Ц



## Ч



## Ш



## Щ



## Э



## Я