ПАВЕЛ АГУРОВ

bhv®

C#

СБОРНИК РЕЦЕПТОВ

Консольные программы

Программы Windows Forms

Базы данных

Работа с аппаратурой

Готовые решения, советы

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Павел Агуров

C#

СБОРНИК РЕЦЕПТОВ

Санкт-Петербург
«БХВ-Петербург»
2007

УДК 681.3.068+800.92C#
ББК 32.973.26-018.1
А27

Агуров П. В.

А27 C#. Сборник рецептов. — СПб.: БХВ-Петербург, 2007. — 432 с.: ил.

ISBN 5-94157-969-1

В книге содержатся советы, алгоритмы и готовые примеры программ из различных областей: шифрование, файловые и сетевые операции, XML, ASP.NET, взаимодействие с MS Office и Internet Explorer и др. Описаны синтаксис языка C#, вопросы отладки и профилирования приложений, а также проблемы, возникающие при переходе с других языков программирования на язык C#. Рассматриваются примеры наиболее часто используемых регулярных выражений. Отдельная глава посвящена работе с аппаратурой. На компакт-диске размещены все исходные коды, приведенные в книге.

Для программистов

УДК 681.3.068+800.92C#
ББК 32.973.26-018.1

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 15.09.06.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 34,83.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 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 5-94157-969-1

Оглавление

Введение

Огромное количество классов Framework, с одной стороны, позволяет выполнять множество операций, которые в других языках программирования требовали установки дополнительных компонентов (только не подумайте, что я говорю, будто бы C# хуже или лучше C++ и Delphi). С другой стороны — это оборачивается необходимостью помнить множество классов и методов. Ничего страшного в этом нет, но мне всегда хотелось минимизировать затраты на поиск стандартных решений.

За несколько лет использования C# у меня накопилось множество небольших программок, решающих каждодневно возникающие задачи. Если мне нужно выполнить проверку XML-файла или обратиться к устройству USB — я просто открываю свой сборник и копирую из него нужный фрагмент кода. Если необходимого примера там нет, я его добавляю. Кроме того, сюда включены некоторые вопросы, которые я задаю на интервью при приеме на работу. Так накопился не очень большой, но, как мне кажется, полезный сборник небольших примеров или просто ответов на часто возникающие вопросы, который я и решился представить вашему вниманию.

Для кого эта книга

Эта книга для программистов-практиков. Не ищите в ней теоретических знаний — для этого есть множество замечательных изданий. Эта книга — набор готовых решений, советов и исходного кода.

Предполагается, что читатель знаком с синтаксисом языка C# и имеет представление об архитектуре платформы .NET.

О программном коде

Код, приведенный в книге, собирался в течение нескольких лет из различных источников. Это и книги, и Интернет, и вопросы интервью... Я старался указывать автора, но, конечно, это не всегда было возможно.

Каждый листинг, приведенный в книге, проверялся и при необходимости корректировался.

Из-за того что книга содержит, по большей части, неполные листинги программ (в большинстве примеров удалена автоматически генерируемая и системная информация), мне пришлось пойти на некоторый компромисс. С одной стороны, использование оператора `using` прибавило бы читабельности кода, но с другой — необходимость каждый раз давать пояснение, что "для работы этого фрагмента кода следует указывать следующие описания...", усложнило бы и без того нелегкую жизнь. Итак, директивы `using` во фрагментах кода не указываются, зато имена классов приводятся полностью.

Полный и компилируемый исходный код содержится на компакт-диске.

Краткое описание глав

Книга состоит из 28 глав, двух приложений и списка литературы.

В *главах 1—12* обсуждаются общие вопросы платформы .NET.

- *Глава 1* описывает общие вопросы, связанные с платформой .NET, установкой Microsoft Visual Studio и установкой Framework.
- *Глава 2* содержит некоторые конструкции C#, которые могут оказаться небесполезными в повседневной работе, а также отвечает на несколько каверзных вопросов, которые часто задаются на интервью.
- *Глава 3* будет интересна тем, кто переходит на C# с других языков программирования.
- *Глава 4* содержит примеры работы с технологией работы с информацией времени исполнения.
- *Глава 5* описывает вопросы, связанные с отладкой кода и возникающими при этом проблемами.
- *Глава 6* содержит информацию обо всем, что связано с командной строкой, конфигурациями и свойствами программ.
- *Глава 7* предлагает примеры всевозможных преобразований, как строки в число, так и кодировки Win1251 в KOI-8 или HTML в обычный текст.
- *Глава 8* содержит информацию о всевозможных операциях с множествами.
- *Глава 9* содержит примеры кода, позволяющего зашифровать строку, вычислить контрольную сумму и т. п.
- *Глава 10* содержит информацию, связанную со строками.
- *Глава 11* предоставляет примеры кода, полезного при написании консольных программ.

- *Глава 12* содержит советы по повышению производительности приложений.

Главы 13—14 посвящены файлам и файловым операциям.

- *Глава 13* содержит примеры работы со списками файлов и каталогов.
- *Глава 14* предлагает примеры работы с файлами и каталогами.

Глава 15 содержит примеры работы с сетью, сетевые операции и работу с Интернетом.

В *главах 16—19* обсуждаются примеры системных операций.

- *Глава 16* описывает работу с ярлыками программ, папкой автозапуска, получение разрешения экрана и т. п.
- *Глава 17* показывает примеры работы с потоками и описывает вопросы синхронизации.
- *Глава 18* содержит примеры запуска внешних процессов, получение списка процессов, управление сервисами и т. д.
- *Глава 19* содержит множество интересных примеров, которым не нашлось место в предыдущих главах.

Глава 20 посвящена регулярным выражениям. Это исключительно мощный инструмент! Об этом надо обязательно знать и уметь им пользоваться.

Глава 21 посвящена работе с XML и всем связанным с ним вопросам.

Глава 22 содержит примеры программ, отвечающих на часто возникающие вопросы при разработке приложений Windows Application.

Глава 23 отвечает на некоторые вопросы, связанные с разработкой баз данных.

Глава 24 содержит решение вопросов, возникающих при разработке приложений ASP.NET.

Глава 25 показывает примеры взаимодействия с MS Office и Internet Explorer.

Глава 26 содержит примеры работы с WMI.

Глава 27 показывает примеры работы с аппаратурой, в частности, с COM- и USB-портами.

Глава 28 демонстрирует примеры работы с изображениями: градиентная заливка, создание скриншотов, преобразование форматов изображений и т. д.

Благодарности

Хочется поблагодарить всех сотрудников компании EPAM (**www.epam.com**), где я имею честь работать уже несколько лет. Работать в компании более по-

лутора тысяч человек действительно интересно, и это не могло не повлиять на наполнение моего сборника.

Благодарю всех сотрудников издательства "БХВ-Петербург", помогавших мне в процессе создания книги, особенно Игоря Шишигина и Евгения Рыбакова.

Обратная связь

Почему-то хочется верить, что в этой книге нет ошибок и опечаток. Но в действительности, это, конечно же, не так. Если вы найдете неточности или ошибки, или просто захотите пообщаться — пишите **mail@bhv.ru**.

ГЛАВА 1

Технология .NET

1.1. На каких платформах запускается .NET?

Для работы программ поддерживаются Windows XP, Windows 2000, NT4 SP6a и Windows ME/98. Операционная система Windows 95 не поддерживается. На некоторых платформах поддерживается не весь набор функциональности, например, ASP.NET работает только на Windows XP и Windows 2000. Операционные системы Windows 98/ME не могут использоваться для разработки. IIS не поддерживается в Windows XP Home Edition, которая также не может выступать в качестве сервера ASP.NET. Однако Web-сервер ASP.NET Web Matrix может работать на Windows XP Home Edition. Проект Mono (**www.go-mono.com**) позволяет работать на Linux. Еще один Framework-проект — DotGNU — представлен на сайте **dotgnu.org**.

1.2. Какие компиляторы существуют для .NET?

Microsoft предлагает компиляторы для C#, C++, VB и JScript. Другие производители предлагают компиляторы для COBOL, Eiffel, Perl, Smalltalk, Python и т. д. Обширный список языков можно найти на странице **www.dotnetpowered.com/languages.aspx**.

1.3. Можно ли запустить программу без .NET Framework?

Нет, нельзя.

Однако компания RemoteSoft (**http://www.remotesoft.com/linker**) выпустила полезный инструмент, который умеет "привязывать" исполняющую среду к программе на .NET.

В Windows Vista ситуация изменится — библиотека .NET будет встроена в операционную систему.

1.4. Можно ли вызывать функции Win32 из .NET-программ?

Да, можно. Для этого надо описать прототип функции с атрибутом `DllImport`. Для работы требуется библиотека `System.Runtime.InteropServices`, как показано в листинге 1.1.

Листинг 1.1. Вызов функции Win32 из .NET-программы

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

namespace MessageBox
{
  class MainApp
  {
    [DllImport("user32.dll", EntryPoint="MessageBox",
         SetLastError=true, CharSet=CharSet.Auto)]
    public static extern int MessageBox(int hWnd, String strMessage,
                              String strCaption, uint uiType);
    public static void Main()
    {
       MessageBox(0, "Вызов функции Win32!", ".NET", 0);
    }
  }
}
```

На сайтах **custom.programming-in.net** и **pinvoke.net** можно найти множество оберток для функций Win32.

1.5. Проблемы установки Framework

При попытке установки пакета Framework появляются странные ошибки, установка обрывается. Одна из вероятных причин — антивирус. Особенно это относится к DrWeb. Попробуйте удалить антивирус из системы, установить Framework и проинсталлировать антивирус снова.

1.6. Ускорение загрузки модулей

С помощью утилиты Ngen.exe можно загрузить модуль в специальный кэш. Это позволит ускорить загрузку и исполнение модуля:

```
ngen [options] [assemblyName | assemblyPath]
```

Например, загрузка в кэш:

```
ngen MyAssembly.dll
```

Удаление из кэша:

```
ngen /delete MyAssembly.dll
```

1.7. Одновременная установка Visual Studio 2003 и 2005

После установки Visual Studio 2005 и APS 2.0 предыдущая версия (2003) не работает. Ошибка выглядит примерно так:

Auto-attach to process '[1312] aspnet_wp.exe' on machine 'EPSW012' failed. Error code 0x8013134b.

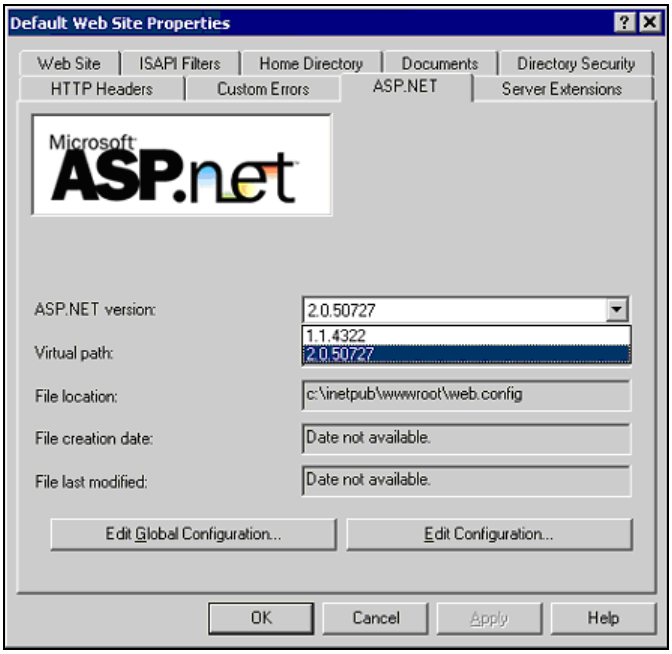

Рис. 1.1. Переключение версии ASP.NET

Для исправления проблемы следует зайти в настройки **Default Web Site в IIS**, выбрать команду **Свойства** в контекстном меню и переключить версию ASP на нужную (рис. 1.1). Если это не помогло, следует выполнить команду:

```
aspnet_regiis.exe -i
```

После установки новой версии Enterprise Manager следует перерегистрировать модуль SQLDMO.DLL.

1.8. Не запускается отладчик в Visual Studio

1.8.1. Ошибка безопасности

При запуске отладчика возникает ошибка:

Error while trying to run project: Unable to start debugging on the web server. You do not have permissions to debug. Verify that you are a member of the 'Debugger Users' group on the sever.

Как минимум, должны быть выполнены следующие условия:

- пользователь, с правами которого производится запуск отладчика, должен входить в группу `Debugger Users`;
- в IIS должна быть включена опция **Integrated Windows Authentication**;
- должна быть выставлена опция `debug="true"` в файле web.config;
- должна быть включена опция **Enable ASP.NET debugging** в свойствах проекта.

Если это не помогает, включите сайт **http://localhost** в список **Trusted Sites**. Второй вариант — попробовать сбросить настройки безопасности Internet Explorer в значения по умолчанию (**Reset IE security settings**).

Ссылки на статьи:

- **www.gotdotnet.com/team/csharp/learn/whitepapers/howtosolvedebuggerproblems.doc**;
- **support.microsoft.com/default.aspx?scid=kb;EN-US;319842**.

1.8.2. Ошибка конфигурирования

Собщение об ошибке имеет вид:

Unable to start debugging on the web server. The project is not configured to be debugged.

Для ее исправления следует открыть конфигурацию сайта в IIS и нажать кнопку **Create**.

Ссылки на статью:

- **www.prezzatech.com/kb/articles/kb-1020-unable_to_start_debuggin_on_web_server.asp**;
- **www.cryer.co.uk/brian/mswinswdev/ms_vbnet_unable_start_debugging.htm**.

1.8.3. Ошибка 401

Сообщение об ошибке имеет вид:

HTTP 401.1 — Unauthorized: Logon Failed.

Для исправления Microsoft советует проверить наличие ключей в реестре в ветке

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa`

Ключ `DisableLoopbackCheck` должен иметь значение 1 (тип `DWORD`). Если такого ключа нет, то его нужно создать.

Ссылка на статью: **support.microsoft.com/default.aspx?scid=kb;en-us;896861**.

1.9. Не запускается отладчик в Visual Studio 2005

При запуске отладчика в Visual Studio 2005 возникает ошибка:

Error while trying to run project: Unable to run debugging. The binding handle is invalid.

Для решения проблемы следует зайти в свойства проекта (**Project | [*ProjectName*] Properties | Debug**) и выключить опцию **Enable the Visual Studio hosting process**. Если проект состоит из нескольких модулей, нужно выполнить эту процедуру для каждого модуля.

Ссылка на статью в MSDN: **lab.msdn.microsoft.com/productfeedback/ViewWorkaround.aspx?FeedbackID=FDBK36247#1**.

ГЛАВА 2

Вопросы синтаксиса C#

2.1. Как выглядит простая C#-программа?

В C# не существует глобальных функций и переменных, поэтому минимальная программа должна содержать хотя бы один класс (для простой программы указание пространства имен не обязательно, хотя среда Visual Studio вставляет это описание по умолчанию):

```
class CApplication
{
  public static void Main()
  {
    System.Console.Write("Минимум C#");
  }
}
```

2.2. Оператор *switch* можно использовать для строк

В C# оператор `switch` может работать со строками так же, как с целыми числами (листинг 2.1). Его использование значительно нагляднее и эффективнее, чем набор операторов сравнения.

Листинг 2.1. Работа с оператором `switch`

```
string command;
.............................
switch (command) {
```

```
  // Вычисление среднего арифметического
  case "MEAN":
  {
    ..............................
    break;
  }
  // Найти минимальное число
  case "MIN":
  {
    ..............................
    break;
  }
  // Найти максимальное число
  case "MAX":
  {
    ..............................
    break;
  }
  default:{
    Console.WriteLine("Неправильная команда!");
    break;
  }
}
```

Однако, в отличие от других языков, C# требует обязательного наличия оператора `break` в конце каждой ветки. Для перехода между ветками используется оператор `goto` *(см. разд. 3.7)*.

2.3. Выходные параметры без инициализации

В C# требуется обязательная инициализация переменных перед любым их использованием. Однако иногда предварительная инициализация переменных выглядит несколько странно:

```
void SomeFunc(ref int out_var)
{
   out_var = 1;
}
..............................
int index= 0;
SomeFunc(ref index);
```

Как видно, требования синтаксиса C# выполнены, но с другой стороны, инициализация параметра, значение которого будет вычислено внутри метода,

выглядит несколько странно. Проблема заключается в том, что компилятор не может знать, будет ли значение этой переменной использоваться в методе до первой инициализации или нет. Для явного обозначения параметра как выходного применяется ключевое слово `out`:

```
void SomeFunc(out int out_var)
{
  // Использование out-параметра без инициализации
  // приведет к ошибке компиляции
  // int test = out_var;

   out_var = 1;
}
.............................
int index;
SomeFunc(out index);
```

Теперь компилятор будет блокировать использование переменной `out_var` внутри метода до ее инициализации.

2.4. Константы и поля "только для чтения"

В стандарте C# предусматриваются константы и поля "только для чтения" (`readonly`):

```
public readonly int ReadonlyValue  = 22;
public const       int ConstValue  = 15;
```

Разница между этими описаниями заключается в том, что константы вычисляются на стадии компиляции, а значения "только для чтения" определяются лишь на стадии выполнения программы. Одним из последствий этого отличия является ситуация, когда библиотека и использующая ее программа компилируются отдельно. Если в библиотеке использовать константу, то при изменении ее значения (и перекомпиляции библиотеки) нужно перекомпилировать и программу. Если же использовать поля "только для чтения", то программу перекомпилировать не надо, т. к. значение поля определяется на стадии исполнения.

Так как `readonly`-поля определяются на стадии выполнения, то следующий код будет нормально компилироваться и работать:

```
class Test
{
  public string Name;
}
```

```
class Class1
{
  public static readonly Test test = new Test();

  [STAThread]
  static void Main(string[] args)
  {
    test.Name = "Program";
  }
}
```

Такая конструкция удобна для создания единственного экземпляра объекта, т. к. присвоение

```
test = new Test(); // ошибка!
```

вызовет ошибку "A static readonly field cannot be assigned to (except in a static constructor or a variable initializer" ("Статические `readonly`-поля не могут быть установлены, исключая статические конструкторы и инициализацию переменных").

2.5. Как определить ссылочный тип?

Определить ссылочный тип можно с помощью метода `IsValueType` (листинг 2.2).

Листинг 2.2. Использование метода `IsValueType`

```
using System;

namespace IsValueType
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      Console.WriteLine(typeof(int).IsValueType);    // true
      Console.WriteLine(typeof(string).IsValueType); // false
      Console.WriteLine(typeof(Enum).IsValueType);   // false
    }
  }
}
```

2.6. Псевдонимы

Оператор `using` можно применять не только для указания используемых библиотек, но и для сокращенной записи некоторых классов. Псевдонимы записываются вне описания класса, но могут указываться внутри пространства имен. Например:

```
using Debug = System.Diagnostics.Debug;
Debug.Assert(index<0, "Сообщение об ошибке", "Подробности");
```

С использованием псевдонимов можно осуществить мечту программистов 70-х годов прошлого века — программу на русском языке[1]:

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

// Описание переменных
целое перем1 = 5;
дробное перем2 = 2.55F;
```

2.7. Что означает символ @ в начале строки?

Знак `@` в начале строки (перед открывающей кавычкой) позволяет записывать строку, не дублируя слеши, как это делалось с C++. Таким образом, строка

```
string fileName = "c:\\temp\\test.txt";
```

полностью идентична строке

```
string fileName = @"c:\temp\test.txt";
```

но вторая выглядит значительно читабельнее.

2.8. Что означает символ @ перед идентификатором?

Символ `@` перед идентификатором означает, что компилятор должен разрешить использовать данное имя идентификатора, даже если это зарезервированное слово:

```
int this = 42;  // не компилируется, ошибка:
                // 'this' is a reserved word
int @this = 42; // компилируется
```

Правда, не очень понятно, зачем это нужно?

---

[1] Согласно спецификации C# для именования объектов программы (классы, методы, переменные и пр.) используются символы Unicode.

2.9. Какая разница между оператором *is* и методом *IsSubclassOf*?

Разница заключается в следующем:

1. Оператор `is` компилируется в инструкцию `IL asclass`, а вызов `IsSubclassOf` — это вызов метода, что, конечно, дольше.
2. Оператор `is` можно использовать с первым операндом, равным `null`, а вызвать метод у такого объекта не получится.
3. Оператор `is` работает и с интерфейсами, а метод только с классами.
4. В конце концов, сама логика работы разная: оператор возвращает `true`, если объект принадлежит к тому же классу или его наследнику, а метод — только к наследнику.

Обойти ограничения пунктов 3 и 4 можно, используя метод `Type.IsAssignableFrom()`, однако вопрос скорости работы все равно остается.

2.10. Какая разница между оператором *as* и прямым приведением типа?

В C# существуют два способа приведения типов: оператор `as` и прямое приведение типа. Отличие этих способов в том, что при невозможности прямого приведения типов будет сгенерировано исключение `InvalidCastException`, тогда как оператор `as` просто вернет `null`. Вполне вероятно, что после получения нулевого указателя от оператора `as`, где-то в коде будет сгенерировано исключение `NullReferenceException` (если получение `null` не предусматривалось логикой программы). Таким образом, использование прямого приведения предпочтительнее, т. к. найти ошибку в этом случае значительно проще, чем искать безымянный `NullReferenceException`.

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

Никакой. В C# определено несколько стандартных имен (табл. 2.1). Вполне допустимо (с точки зрения синтаксиса) смешанная запись, например:

```
string str= new System.String(' ', 5);
```

Таблица 2.1. *Стандартные имена типов C#*

Короткое имя	CLR-тип
`string`	`System.String`
`sbyte`	`System.SByte`

Таблица 2.1 *(окончание)*

Короткое имя	CLR-тип
byte	System.Byte
short	System.Int16
ushort	System.UInt16
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal

2.12. Какая разница между *override* и *new*?

Ключевое слово `override` означает перекрытие виртуального метода, а `new` — просто скрытие. Эту разницу демонстрирует листинг 2.3.

Листинг 2.3. Различие между `override` и `new`

```
using System;

namespace VirtualNewMethods
{
  class Base
  {
    public virtual void Test()
    {
      Console.WriteLine("Base::Test");
    }
  }

  class Derived1 : Base
  {
    public override void Test()
```

```
    {
      Console.WriteLine("Derived1::Test");
    }
  }

  class Derived2 : Base
  {
    public new void Test()
    {
      Console.WriteLine("Derived2::Test");
    }
  }

  class MainClass
  {
    [STAThread]
    static void Main(string[] args)
    {
      Derived1 d1 = new Derived1();
      d1.Test(); // Напечатает "Derived1::Test"

      Derived2 d2 = new Derived2();
      d2.Test(); // Напечатает "Derived2::Test"

      Base b1 = new Derived1();
      b1.Test(); // Напечатает "Derived1::Test"

      Base b2 = new Derived2();
      b2.Test(); // Напечатает "Base::Test" (!)
    }
  }
}
```

2.13. Какая разница между == и *Equals*?

Разницу между операторами == и `Equals` показывает листинг 2.4.

Листинг 2.4. Различие между == и `Equals`

```
// Строки
string a = new string(new char[] {'h', 'e', 'l', 'l', 'o'});
string b = new string(new char[] {'h', 'e', 'l', 'l', 'o'});
```

```
Console.WriteLine (a==b);        // Вернет true
Console.WriteLine (a.Equals(b)); // Вернет true

// Объекты
object c = new string(new char[] {'h', 'e', 'l', 'l', 'o'});
object d = new string(new char[] {'h', 'e', 'l', 'l', 'o'});

Console.WriteLine (c==d);        // Вернет false
Console.WriteLine (c.Equals(d)); // Вернет true
```

2.14. Перекрытие операторов

При перекрытии оператора сравнения нужно следить, чтобы не возник рекурсивный вызов. В листинге 2.5 представлен пример перекрытия операторов.

Листинг 2.5. Перекрытие операторов

```
using System;

namespace OverrideOperation
{

  class Fraction
  {
    private int numerator;
    private int denominator;

    // Конструктор
    public Fraction(int numerator, int denominator)
    {
      this.numerator  = numerator;
      this.denominator= denominator;
    }

    // Сумма
    public static Fraction operator+(Fraction lhs, Fraction rhs)
    {
      // Если делитель одинаковый, можно просто сложить
      if (lhs.denominator == rhs.denominator)
      {
        return new Fraction(
          lhs.numerator+rhs.numerator,
          lhs.denominator
          );
      }
```

```
  // Вычисляем по правилам арифметики
  // 1/2 + 3/4 == (1*4) + (3*2) / (2*4) == 10/8
  int firstProduct  = lhs.numerator * rhs.denominator;
  int secondProduct = rhs.numerator * lhs.denominator;
  return new Fraction(
    firstProduct    + secondProduct,
    lhs.denominator * rhs.denominator
    );
}

// Сравнение на равенство
public static bool operator==(Fraction lhs, Fraction rhs)
{
  // Если надо проверить на null, то нельзя использовать
  // запись (lsh == null), т. к. это вызовет рекурсию.
  // Для проверки на null следует использовать метод Equals

  return
    (lhs.denominator == rhs.denominator) &&
    (lhs.numerator   == rhs.numerator  );
}
// "Не равно" реализуем через "равно"
public static bool operator !=(Fraction lhs, Fraction rhs)
{
  return !(lhs==rhs);
}

// Метод сравнения. Проверяет еще и тип аргумента
public override bool Equals(object o)
{
  if (! (o is Fraction) )
  {
    return false;
  }
  return this == (Fraction) o;
}

// Метод преобразования в строку
public override string ToString()
{
  return string.Format("{0}/{1}", numerator, denominator);
}
```

```
    // Перекрытие == требует реализации GetHashCode
    public override int GetHashCode()
    {
      return base.GetHashCode();
    }
  }

  public class OperatorOverrideClass
  {
    static void Main()
    {
      Fraction f1 = new Fraction(3, 4);
      Console.WriteLine("f1: {0}", f1.ToString());

      Fraction f2 = new Fraction(2, 4);
      Console.WriteLine("f2: {0}", f2.ToString());

      Fraction f3 = f1 + f2;
      Console.WriteLine("f1 + f2 = f3: {0}", f3.ToString());

      Fraction f4 = new Fraction(5, 4);
      if (f4 == f3)
      {
        Console.WriteLine("f4: {0} == F3: {1}",
          f4.ToString(), f3.ToString());
      }

      if (f4 != f2)
      {
        Console.WriteLine("f4: {0} != F2: {1}",
          f4.ToString(), f2.ToString());
      }

      if (f4.Equals(f3))
      {
        Console.WriteLine("{0}.Equals({1})",
          f4.ToString(), f3.ToString());
      }

      Console.ReadLine();
    }
  }
}
```

2.15. Как создать наследника, если конструктор класса имеет параметры?

Если конструктор класса имеет параметры, то наследование производится с помощью ключевого слова `base()`:

```
public class BaseClass
{
  public BaseClass(int x)
  {
     Console.WriteLine("BaseClass constructor: x={0}", x);
  }
}

public class DeriveClass : BaseClass
{
  public DeriveClass(int x) : base(x)
  {
    Console.WriteLine("DeriveClass constructor: x={0}", x);
  }
}
// Выведет:
//    BaseClass constructor  : x=5
//    DeriveClass constructor: x=5
DeriveClass d = new DeriveClass(5);
```

Конструкторы наследников могут не иметь параметров:

```
public class DeriveClass : BaseClass
{
  public DeriveClass() : base(5)
  {
    Console.WriteLine("DeriveClass constructor");
  }
}
```

Или иметь более сложные конструкторы:

```
public class DeriveClass : BaseClass
{
  public DeriveClass(int x) : base(x+10)
 {
    Console.WriteLine("DeriveClass constructor: x={0}", x);
  }
}
```

Последний конструктор при создании выведет строки:

```
BaseClass constructor: x=15
DeriveClass constructor: x=5
```

Конечно, параметры конструкторов не обязательно должны совпадать:

```
public class DeriveClass : BaseClass
{
  public DeriveClass(string s) : base(5)
  {
    Console.WriteLine("DeriveClass constructor: s={0}", s);
  }
}
```

2.16. Что такое делегат?

Пример использования делегата показан в листинге 2.6. Класс `Calculate` производит некоторые вычисления, а вызывающий класс передает делегат для отображения результатов.

Листинг 2.6. Использование делегатов

```
using System;

namespace DelegateExample
{
  // Тип делегата
  public delegate void PrintDelegate(int x , int y);

  class Calculate
  {
    // Сохраняет ссылку на делегат
    private PrintDelegate print;

    public Calculate(PrintDelegate print)
    {
      this.print = print;
    }
    public void Execute()
    {
      for (int i=0; i<10; i++)
      {
        int result = i * i;
```

```
        // Использование делегата
        if (print != null)
          print(i, result);
      }
    }
  }

  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      Calculate c = new Calculate(new PrintDelegate(Print));
      c.Execute();
    }
    // Реальный метод
    static void Print(int x, int y)
    {
      Console.WriteLine("x={0} y={1}", x, y);
    }
  }
}
```

2.17. В чем разница между полем и свойством?

С точки зрения доступа обычное поле и свойство с доступом `get/set` не отличаются (правда, в C# 2.0 можно указывать разные уровни доступа для `get` и `set`):

```
class Test
{
  public int Var1;

  private int _Var2;
  public int Var2
  {
    private get {return(_Var2);}
    set {_Var2 = value;}
  }
}
```

Применение того или иного способа зависит от контекста программы.

- Использование свойства необходимо, если при установке значения свойства требуется выполнить некоторые действия, например, вызвать некий делегат для уведомления об изменении:

```
set {_Var2=value; DoPropertyChangeEvent(); }
```

- Использование свойства необходимо, если при установке значения свойства нужно выполнить некоторые проверки на допустимость свойства, например:

```
set {if (value>0) _Var2=value; }
```

- Использование свойства необходимо, если значение свойства не хранится в самом классе, а, например, читается из базы данных:

```
get { return(ReadFromDB("Var2")); }
set { WriteToDB("Var2", value) }
```

- Если необходим доступ к полю "только для чтения", то можно либо определить свойство с доступом только `get`, либо описать обычное поле как `readonly`.

- Ну и, наконец, описание поля как свойства выдерживает объектно-ориентированный подход: при изменении внутреннего содержания класса внешние интерфейсы класса не изменяются, т. е. при изменении метода работы с полем `Var2` другие классы будут видеть то же самое свойство `Var2`.

2.18. Свойство с индексатором

Свойства с индексатором позволяют обращаться с объектом класса как с массивом, что повышает читабельность кода (листинг 2.7).

Листинг 2.7. Свойство с индексатором

```
using System;

namespace IndexProperty
{
  class TestClass
  {
    public string this[int index]
    {
      get
```

```
      {
        return string.Format("Index={0}", index);
      }
    }

    public string this[string name]
    {
      get
      {
        return string.Format("Name={0}", name);
      }
    }

    public string this[string name, int index]
    {
      get
      {
        return string.Format("Name={0} index={1}", name, index);
      }
    }
  }

  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      TestClass tc = new TestClass();
      Console.WriteLine("{0},{1},{2}", tc[1], tc[2], tc[3]);
      Console.WriteLine("{0},{1},{2}", tc["A"], tc["B"], tc["C"]);
      Console.WriteLine("{0},{1},{2}", tc["A", 1],
                                       tc["B", 2], tc["C", 3]);
    }
  }
}
```

2.19. Можно ли сделать свойство с разным уровнем доступа для *get* и *set*?

В .NET 1.1 нельзя указывать ключевые слова `private` или `public` перед `get` и `set`. Уровень доступа к методам `get` и `set` задается уровнем доступа к самому свойству. Эта проблема решена в .NET 2.0.

2.20. Как вызвать методы с одинаковыми именами из интерфейсов?

Если класс реализует несколько интерфейсов с методами, имеющими одинаковое имя, то вызвать такой метод можно с помощью приведения типа (листинг 2.8).

Листинг 2.8. Вызов методов с одинаковыми именами из интерфейсов

```
using System;

namespace Interfaces
{
  interface I1
  {
    void PrintName();
  }

  interface I2
  {
    void PrintName();
  }

  class BaseTest
  {
    public void PrintName()
    {
      Console.WriteLine("BaseTest.GetName");
    }
  }

  class Test : BaseTest, I1, I2
  {
    void I1.PrintName()
    {
      Console.WriteLine("I1.PrintName");
    }

    void I2.PrintName()
    {
      Console.WriteLine("I2.PrintName");
    }
  }
```

```
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      Test t = new Test();
      t.PrintName();         // вызовет base-метод
      (t as I1).PrintName(); // вызовет I1-метод
      (t as I2).PrintName(); // вызовет I2-метод
    }
  }
}
```

2.21. В чем разница *byte* и *sbyte*?

Для доступа к массивам как к "сырым" блокам памяти (т. е. без интерпретации их как значений конкретных типов) существует класс `System.Buffer`. В данном случае нам поможет метод `BlockCopy`:

```
byte[] src = new byte[100];
sbyte[] dest = new sbyte[100];
Buffer.BlockCopy(src, 0, dest, 0, src.Length);
```

Обратите внимание, что последний параметр указывает размер копируемого блока в байтах, а не в элементах массива. В данном случае (поскольку элементы массива — байты) эти размеры совпадают. Однако при использовании более сложных типов надо действовать осторожнее. Для выяснения размера массива в байтах можно вызвать метод `Buffer.ByteLength()`.

2.22. Как сгенерировать документацию по коду?

Нужно использовать опцию компилятора `/doc:`*<имя_файла>*. В этом случае компилятор сам сгенерирует XML-файл с документацией к вашей сборке на основе ваших тегов. На основе этого файла также можно генерировать CHM- или HTML-описания (в состав Visual Studio 2003 включен инструмент для генерирования HTML-документации).

Если вы используете Visual Studio, укажите в свойствах проекта, в поле **Configuration Properties | Build | XML Documentation File**, строку *<имя_сборки>*`.xml`. Это заставит среду генерировать XML-файл с описанием.

2.23. Как инициализировать поля класса до вызова конструктора?

Для инициализации полей класса можно использовать `inline`-инициализацию:

```
class Test
{
   // inline-инициализация
   ArrayList array = new ArrayList();

   public Test()
   {
       // Поле array уже инициализировано
       array.Add("1");
   }
}
```

2.24. Использование переменного числа параметров

Для передачи переменного числа аргументов служит ключевое слово `params` (в Visual Basic использовалось ключевое слово `ParamArray`). Разумеется, аргумент метода, описывающий переменное число параметров, должен быть последним в списке аргументов. Пример вычисления суммы переменного числа слагаемых приведен в листинге 2.9.

Листинг 2.9. Использование переменного числа параметров

```
using System;

namespace Params
{
  class Class1
  {
    static int Sum(params int[] numbers)
    {
      int total = 0;
      foreach(int i in numbers)
      {
        total += i;
      }
```

```
      return total;
    }

    [STAThread]
    static void Main(string[] args)
    {
      Console.WriteLine(Sum(1,2,3));
      Console.WriteLine(Sum(10,20,30,40,50,60));
    }
  }
}
```

2.25. Указатели

Использование *указателей* в C# возможно, но не рекомендуется. Для работы с указателями необходимо установить опцию компиляции проекта **Allow Unsafe Code Blocks** в значение `True` (рис. 2.1). Пример работы с указателями представлен в листинге 2.10. Еще один пример приводится в разделе работы со строками *(см. разд. 10.9)*.

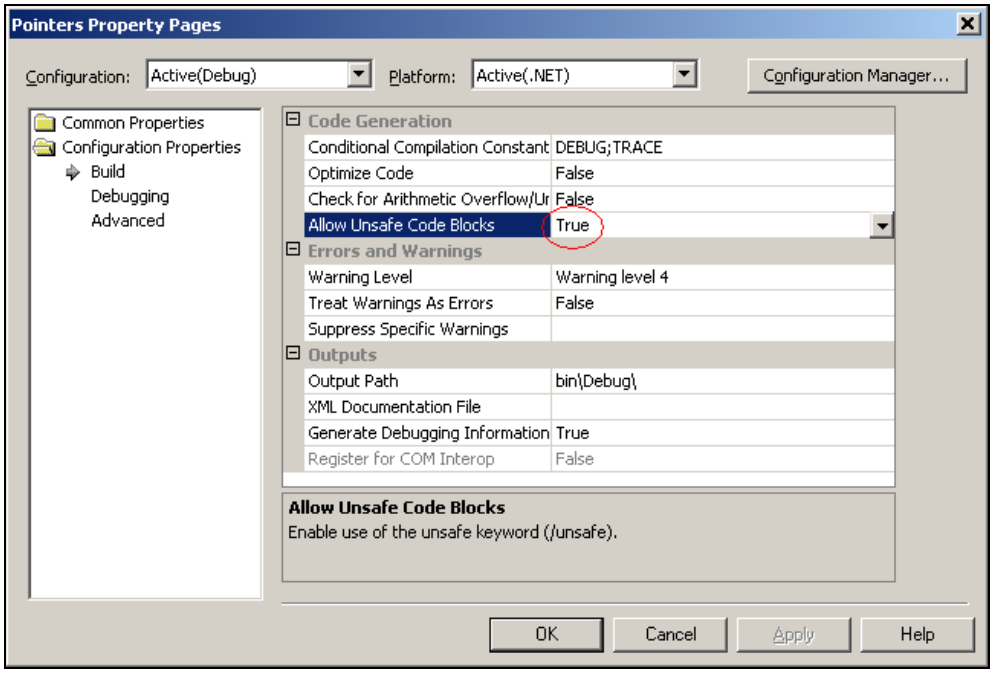

Рис. 2.1. Опция **Allow Unsafe Code Blocks**

Листинг 2.10. Использование указателей

```
using System;

namespace Pointers
{
  class Class1
  {
    unsafe static void TestPointer()
    {
      int *p;   // указатель
      int i;    // переменная

      i = 10;   // инициализация переменной
      p = &i;   // указатель на переменную


      // Выводим значение переменной напрямую и через указатель
      Console.WriteLine(i);
      Console.WriteLine(*p);
      Console.WriteLine("----------");

      // Присваиваем значение переменной через указатель
      *p = 333;
      Console.WriteLine(i);
      Console.WriteLine(*p);
      Console.WriteLine("----------");

      // Изменяем переменную через собственный указатель
      i = *(&i) + 10;
      Console.WriteLine(i);
      Console.WriteLine(*p);
    }

    [STAThread]
    static void Main(string[] args)
    {
      TestPointer();
    }
  }
}
```

2.26. Слабые ссылки

Слабые ссылки (weak reference) используются при необходимости хранения в памяти необязательных объектов большого размера. Например, получив с SQL-сервера данные, программа может хранить их в памяти до тех пор, пока эта память не потребуется системе для своих нужд. Сохранение данных через обычную (сильную) ссылку не дает возможности освобождения памяти. Именно в этом случае пригодятся слабые ссылки. В листинге 2.11 приведен пример работы со слабыми ссылками.

Листинг 2.11. Использование слабых ссылок

```
using System;

namespace WeakReferenceTest
{
  class TestClass
  {
    public TestClass()
    {
      Console.WriteLine("Создание объекта TestClass");
    }

    public void Print()
    {
      Console.WriteLine("Метод Print");
    }
  }

  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Сильная ссылка
      TestClass tc = new TestClass();
      // Слабая ссылка
      WeakReference wr = new WeakReference(tc);

      // Вызываем сборщик мусора
      GC.Collect();
      GC.WaitForPendingFinalizers();
      GC.Collect();
```

```
      // Объект не уничтожится, т. к.
      // есть сильная ссылка
      if (wr.IsAlive)
        (wr.Target as TestClass).Print();
      else
        Console.WriteLine("Объект уничтожен");

      // Уничтожаем сильную ссылку
      tc = null;

      // Вызываем сборщик мусора
      GC.Collect();
      GC.WaitForPendingFinalizers();
      GC.Collect();

      // Объект уничтожится
      if (wr.IsAlive)
        (wr.Target as TestClass).Print();
      else
        Console.WriteLine("Объект уничтожен");

    }
  }
}
```

2.27. Файлы ресурсов

При создании приложений, работающих с разными языками, удобно собрать все строки, требующие перевода, в едином файле. Для добавления такого ресурсного файла нужно выбрать меню **Add New Item** и добавить файл типа **Assembly Resource File**.

Добавлять можно не только строки, но и ресурсы других типов, например, `System.Int32`.

С помощью класса `System.Resources.ResourceManager` можно загружать ресурсные файлы, соответствующие различным культурам (листинг 2.12).

Листинг 2.12. Использование файла ресурсов

```
using System;
using System.Reflection;
using System.Resources;
using System.IO;
using System.Globalization;
```

```
namespace Resource
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Имя ресурсного файла (файлов)
      string [] resNames =
          Assembly.GetCallingAssembly().GetManifestResourceNames();
      resource = resNames[0];

      // Загружаем
      string baseName =
                  resource.Substring(0, resource.LastIndexOf('.'));
      ResourceManager resourceManager = new ResourceManager(
                  baseName, Assembly.GetExecutingAssembly());

      // Hashtable ресурсов
      ResourceSet resourceSet = resourceManager.GetResourceSet(
                  CultureInfo.CurrentCulture, true, true);

      // Доступ к ресурсам
      // Строка
      Console.WriteLine(resourceSet.GetString("Str1"));
      // Тип System.Int32
      Console.WriteLine(resourceSet.GetObject("Int1").GetType());
      Console.WriteLine((Int32)resourceSet.GetObject("Int1"));
    }
  }
}
```

ГЛАВА 3

Для программистов C++ и Delphi

3.1. Как удалить объекты, созданные с помощью оператора *new*?

Никак. Они сами удалятся сборщиком мусора.

3.2. Как создать экземпляр класса на стеке?

Никак. Все классы создаются в куче и контролируются сборщиком мусора.

3.3. Почему не вызывается деструктор?

В C# деструктором является реализация метода `Finalize()`, однако никакой гарантии его вызова нет. Единственная гарантия — деструктор будет вызван при закрытии программы *(см. также разд. 9.12)*.

3.4. Почему в операторе *if* больше нельзя использовать условия типа *int*?

В C++ в качестве условия оператора `if` можно было использовать выражения типа `int` и им аналогичные. К чему это приводило? С одной стороны, это, конечно, удобно. Но, с другой, оператор `==` (двойное "равно"), требуемый для операции сравнения, легко превращался в оператор присваивания:

```
int x;
if (x == 0) // проверка x на равенство 0
{
}
```

```
if (x = 5) // всегда вернет true и присвоит x=5
{
}
```

Найти такую ошибку очень сложно. В C# такая конструкция недопустима — требуется явное указание логического условия. Может быть, написание нескольких дополнительных символов кому-то покажется лишней работой, но время, потраченное на выявления банальной опечатки, значительно дороже.

3.5. Как создать массив констант?

Действительно, конструкции типа

```
const int [] constIntArray = newint [] {2, 3, 4};
const int [] constIntArrayAnother = {2, 3, 4};
```

вызывают ошибку компиляции. Выходов два — либо использовать описатель `readonly`:

```
static readonly int [] constIntArray = new int[] {1, 2, 3};
```

либо использовать тип `enum` и именованные значения констант.

3.6. Как инициализировать массив структур?

Пример статической инициализации массива структур показан в листинге 3.1.

Листинг 3.1. Инициализация массива структур

```
using System;

namespace StaticStructArray
{
  public struct MyStruct {
    public MyStruct(int x, int y) {
      a = x;
      b = y;
    }
    int a;
    public int A { get {return a;}}
    int b;
    public int B { get {return B;}}
  };
```

```
  class Class1
  {
    static MyStruct [] s = {new MyStruct(5,5), new MyStruct(10,10)};

    [STAThread]
    static void Main(string[] args)
    {
      Console.WriteLine(s[0].A);
    }
  }
}
```

3.7. Оператор *switch* работает не так, как в C++

В C++ допускалось не указывать оператор `break` в секциях `case`:

```
switch (x)
{
  case 0:
    Console.WriteLine(x);  // выполнится, если x==0
  case 1:
    Console.WriteLine(x);  // выполнится, если x==1 или 0
  default:
    Console.WriteLine(x);  // выполнится всегда
    break;
}
```

Этой возможностью пользовались не многие, зато число ошибок, возникающих по забывчивости, исчислению не поддается. Разработчики C# позаботились о том, чтобы эта конструкция больше не компилировалась. При необходимости программист должен явно указать переход на другую ветку с помощью оператора `goto`:

```
switch (x)
{
  case 0:
    // что-то делаем
    goto case 1;
  case 1:
    // что-то делаем
    goto default;
  default:
   // что-нибудь делаем
   break;
}
```

3.8. Есть ли в C# макросы и препроцессор?

Макросов в понимании C++ в C# нет. От препроцессора остались только конструкции:

- `#define` — определяет символ для `#if`, но не макроподстановку;
- `#undef` — удаляет символ;
- `#if`, `#elif`, `#else` и `#endif` — условная компиляция;
- `#error`, `#warning`, `#line` — используется для генерации ошибок;
- `#region`, `#endregion` — используется для выделения секций в исходном коде.

Для условной компиляции можно применять также атрибут `Condition` *(см. разд. 5.1)*.

3.9. Как описать битовые поля?

В C# для работы с битовыми полями предусмотрен класс `BitVector32` (листинг 3.2). Этот класс более эффективен, чем `BitArray` *(см. разд. 8.10)*.

Листинг 3.2. Использование класса `BitVector32`

```
using System;
using System.Collections.Specialized;

namespace BitVector32Test
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      BitVector32 vector1 = new BitVector32( 0 );

      // Маски для первых пяти битов
      int bit1 = BitVector32.CreateMask();
      int bit2 = BitVector32.CreateMask(bit1);
      int bit3 = BitVector32.CreateMask(bit2);
      int bit4 = BitVector32.CreateMask(bit3);
      int bit5 = BitVector32.CreateMask(bit4);

      // Получаем BitVector32{00000000000000000000000000000000}
      Console.WriteLine(vector1.ToString());
```

```
      vector1[bit1] = true; // установить 1-й бит
      // Получаем BitVector32{00000000000000000000000000000001}
      Console.WriteLine(vector1.ToString());
      vector1[bit3] = true; // установить 3-й бит
      // Получаем BitVector32{00000000000000000000000000000101}
      Console.WriteLine(vector1.ToString());
      vector1[bit5] = true; // установить 5-й бит
      // Получаем BitVector32{00000000000000000000000000010101}
      Console.WriteLine(vector1.ToString());

      BitVector32 vector2 = new BitVector32( 0 );

      // Создаем 4 секции. Первая имеет максимум
      // значений — 6, т. е. занимает 4 бита. Вторая - максимум 3
      // (т. е. два бита), затем 1 бит, затем 4 бита.
      BitVector32.Section sect1 = BitVector32.CreateSection( 6 );
      BitVector32.Section sect2 =
                      BitVector32.CreateSection( 3, sect1 );
      BitVector32.Section sect3 =
                      BitVector32.CreateSection( 1, sect2 );
      BitVector32.Section sect4 =
                      BitVector32.CreateSection(15, sect3 );

      vector2[sect2] = 3;
      // Получим BitVector32{00000000000000000000000000011000}
      Console.WriteLine(vector2.ToString());
      vector2[sect4] = 1;
      // Получим BitVector32{00000000000000000000000001011000}
      Console.WriteLine(vector2.ToString());
    }
  }
}
```

ГЛАВА 4

Отражение и связанные с ним вопросы

4.1. Создание объекта по имени типа

Для создания объекта по имени типа используются методы отражения (reflection) (листинг 4.1). Если класс создаваемого объекта находится в том же пространстве имен, что и создающий код, то можно использовать простое создание экземпляра объекта по его типу. Такой способ удобен, если описание набора объектов хранится в некотором конфигурационном файле (например, XML). Создание объектов по имени типа позволяет избавиться от оператора `switch`, создающего нужный объект в зависимости от имени.

Если класс имеет несколько конструкторов или конструктор с параметрами, то необходимо указать, какой именно конструктор нужно найти (листинг 4.2).

Листинг 4.1. Создание объекта по имени типа

```
using System;

namespace Reflection1
{
  public class TestClass
  {
    public TestClass()
    {
       Console.WriteLine("Конструктор TestClass");
    }
  }
```

```
  class ReflectionDemoClass
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Получить ссылку на тип по имени
      Type TestType =
                    Type.GetType("Receipt.TestClass", false, true);
      // Если класс найден
      if (TestType != null)
      {
        // Получаем конструктор
        System.Reflection.ConstructorInfo ci =
                          TestType.GetConstructor(new Type[]{});
        // Вызываем конструктор
        object Obj = ci.Invoke(new object[]{});
      }
      else
      {
        Console.WriteLine("Класс не найден");
      }
    }
  }
}
```

Листинг 4.2. Поиск конструктора с параметрами

```
using System;
using System.Reflection;

namespace FindCustomContructor
{
  class TestClass
  {
    TestClass()
    {
    }

    TestClass(int x, string str)
    {
    }
  }
```

```
  class MainClass
  {
    [STAThread]
    static void Main(string[] args)
    {
      Type type = typeof(TestClass);

      // Пробуем найти обычный конструктор
      ConstructorInfo ci1 = type.GetConstructor(new Type[]{});
      Console.WriteLine("Конструктор 1 "+
                        ((ci1 == null)?"найден":"не найден"));

      // Пробуем найти конструктор с параметрами
      ConstructorInfo ci2 = type.GetConstructor(
                         new Type[]{typeof(int), typeof(string)});
      Console.WriteLine("Конструктор 2 "+
                       ((ci2 == null)?"найден":"не найден"));
    }
  }
}
```

4.2. Вызов метода по имени

Вызвать метод по имени можно с помощью отражения (листинг 4.3). Можно даже вызвать `private`-метод, указав соответствующие флаги поиска:

```
Type someType = someInstance.GetType();
MethodInfo mi = someType.GetMethod(
         "System.Windows.Forms.ISomeInterface.SomeMethod",
          BindingFlags.NonPublic | BindingFlags.Instance
  );
if(mi != null)
{
  mi.Invoke(someInsance, new object[]{});
}
```

Листинг 4.3. Вызов метода по имени

```
using System;
using System.Reflection;

namespace FindMethod
{
  class Test
```

```
  {
    string name;

    Test (string name)
    {
      this.name = name;
    }

    public void ShowName()
    {
      Console.WriteLine(name);
    }

    static void Main()
    {
      // Создать экземпляр объекта
      Test t = new Test ("The name");
      // Получить метод, который хотим вызвать
      MethodInfo mi = typeof(Test).GetMethod("ShowName");
      // Вызвать
      mi.Invoke (t, null);
    }
  }
}
```

4.3. Загрузка и создание класса из другой сборки

Для создания класса из другой сборки (модуля) можно использовать класс `System.Activator`, например:

```
// Создаем объект из сборки Plugins1.exe или
// Plugins1.dll (при загрузке расширение не указывается).
// Пространство имен внутри сборки имеет имя PluginNamespace,
// а сам класс называется TestClass
object Obj =
  Activator.CreateInstance("Plugins1", "PluginNamespace.TestClass");
```

4.4. Получение списка типов из сборки

Перечисление всех доступных типов сборки можно осуществить с помощью вызова метода `GetTypes` (листинг 4.4).

Листинг 4.4. Получение списка типов из сборки

```
using System;

namespace Reflection2
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Загрузить модуль
      System.Reflection.Assembly assembly =
           System.Reflection.Assembly.LoadFrom("Reflection1.exe");
      // Получить список всех классов модуля
      Type[] types = assembly.GetTypes();
      // Отобразить список классов модуля
      foreach (Type t in types)
      {
        Console.WriteLine(t.FullName);
      }
      object Obj = Activator.CreateInstance(
                "Reflection1", "Reflection1.TestClass");
    }
  }
}
```

4.5. Получение всех свойств класса

Получение свойств класса таково:

```
System.Reflection.PropertyInfo[] Props =
                                listBox1.GetType().GetProperties();
foreach(System.Reflection.PropertyInfo prop in Props)
{
  listBox1.Items.Add(prop.PropertyType.ToString()+ ":"+ prop.Name);
}
```

4.6. Установка значения *private*-поля

С помощью методов отражения можно устанавливать значения даже `private`- и `protected`-полей (листинг 4.5).

Листинг 4.5. Установка значения `private`- и `protected`-полей

```
using System;
using System.Reflection;

namespace Reflection3
{
  class Class1
  {
    public class Account
    {
      private    long privatevalue = 0;
      protected  long protectedval = 0;

      public long GetPrivatevalue()
      {
        return privatevalue;
      }

      public long GetProtectedval()
      {
        return protectedval;
      }
    }


    [STAThread]
    static void Main(string[] args)
    {
      Account a = new Account();
      Console.WriteLine("a.privatevalue={0}", a.GetPrivatevalue());
      Console.WriteLine("a.protectedval={0}", a.GetProtectedval());

      FieldInfo fi1 = a.GetType().GetField("privatevalue",
                  BindingFlags.NonPublic | BindingFlags.Instance);
      if (fi1 != null)
      {
        fi1.SetValue(a, 1);
      }

      FieldInfo fi2 = a.GetType().GetField("protectedval",
                  BindingFlags.NonPublic | BindingFlags.Instance);
      if (fi2 != null)
```

```
      {
        fi2.SetValue(a, 2);
      }

      Console.WriteLine("a.privatevalue={0}", a.GetPrivatevalue());
      Console.WriteLine("a.protectedval={0}", a.GetProtectedval());
    }
  }
}
```

4.7. Получение списка полей структуры

Получение списка полей структуры показано в листинге 4.6.

Листинг 4.6. Получение списка полей структуры

```
using System;
using System.Reflection;

namespace Reflection4
{
  class Class1
  {
     public struct Account
     {
       public string   FirstName;
       public string   LastName;
       public int      Age;
       public string   Login;
       public string   Password;

      private long     HashCode;
      private string   Passport;
    }

    [STAThread]
    static void Main(string[] args)
    {
      Console.WriteLine("Public:");
      foreach(FieldInfo fi in typeof(Account).GetFields() )
      {
         Console.WriteLine("\t"+ fi.Name );
      }
```

```
      Console.WriteLine("NonPublic:");
      foreach(FieldInfo fi in
                  typeof(Account).GetFields(BindingFlags.NonPublic |
                  BindingFlags.Instance) )
      {
        Console.WriteLine("\t"+ fi.Name );
      }
    }
  }
}
```

4.8. Получение полного имени сборки и ее свойств

Получение имени сборки и ее свойств можно выполнить с помощью метода, приведенного в листинге 4.7.

Листинг 4.7. Получение полного имени сборки и ее свойств

```
using System;
using System.Reflection;

namespace LoadWithPartialName
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      string Name = "System.Windows.Forms";

      string FullName =
        Assembly.LoadWithPartialName(Name).FullName;
      // Напечатает
      // "System.Windows.Forms, Version=1.0.5000.0,
      //      Culture=neutral, PublicKeyToken=b77a5c561934e089"
      Console.WriteLine(FullName);
      Console.WriteLine();

      string Location =
        Assembly.LoadWithPartialName(Name).Location;
      Console.WriteLine(Location);
```

```
      // Напечатает
      // "f:\winxp\assembly\gac\system.windows.forms\
      //    1.0.5000.0__b77a5c561934e089\system.windows.forms.dll"
      Console.WriteLine();

      string ImageRuntimeVersion =
        Assembly.LoadWithPartialName(Name).ImageRuntimeVersion;
      // Напечатает
      // "v1.1.4322"
      Console.WriteLine(ImageRuntimeVersion);

      Console.ReadLine();
    }
  }
}
```

4.9. Компиляция кода из программы

Для компиляции кода в IL можно использовать классы пространства имен `System.CodeDom.Compiler`, в частности класс `CodeCompiler`. Этот класс является абстрактным. Его наследники реализуют функциональность компилятора для конкретного языка. В составе .NET Framework поставляются три компилятора: C#, VB.NET и JScript. Чтобы получить экземпляр такого класса, нужно обратиться к методам класса `CodeDomProvider`:

```
CodeDomProvider provider =
                    new Microsoft.CSharp.CSharpCodeProvider();
CodeCompiler compiler = provider.CreateCompiler();
```

После этого можно скомпилировать код в сборку:

```
CompilerParameters parameters = new CompilerParameters();
CompilerResults result = compiler.CompileAssemblyFromFile(
                  parameters, @"C:\MyClass.cs");
Assembly resultAssembly = result.CompiledAssembly;
```

Созданная сборка теперь почти ничем не отличается от сборок, полученных "нормальным" путем (т. е. через Visual Studio или компилятором командной строки). В частности, можно создавать и использовать определенные в ней классы:

```
Type myType = resultAssembly.GetType("MyNamespace.MyClass");
object myClassInstance = Activator.CreateInstance(myType);
```

4.10. Перехват загрузки типов и модулей

Иногда возникает задача перехватить стандартную загрузку модулей и загрузить модуль, лежащий в определенной папке. Например, при решении проблемы локализации приложений ресурсы могут размещаться в папках с именами языка, и в зависимости от выбранного языка нужно загружать соответствующий модуль. Аналогичная задача может возникнуть при поиске модуля, содержащего нужный тип.

Событие `AppDomain.CurrentDomain.TypeResolve` вызывается, если системе (точнее говоря, текущему домену) не удалось найти сборку, содержащую запрошенный тип. Событие `AppDomain.CurrentDomain.AssemblyResolve` вызывается, если системе не удалось найти запрошенную сборку.

Листинг 4.8 содержит пример использования этих событий. Главный модуль `MainClass` использует две сборки: `ClassLibrary1`, содержащую класс `Class1`, и `ClassLibrary2`, содержащую класс `Class2`. Главный модуль не имеет ссылки на первую сборку, поэтому при попытке найти тип `Class1` будет вызываться обработчик `TypeResolve`, в котором мы загрузим нужную сборку самостоятельно. На вторую сборку ссылка имеется, но у нее выставлен тип `CopyLocal=False`, поэтому система не сможет определить местонахождение сборки и вызовет обработчик `AssemblyResolve`, в котором мы загрузим эту сборку самостоятельно.

Листинг 4.8. Обработка событий `AssemblyResolve` и `TypeResolve`

```
using System;
using System.Reflection;

namespace ResolveAssembly
{
  class MainClass
  {
    static MainClass()
    {
      AppDomain.CurrentDomain.TypeResolve +=
           new ResolveEventHandler(CurrentDomain_TypeResolve);
      AppDomain.CurrentDomain.AssemblyResolve +=
           new ResolveEventHandler(currentDomain_AssemblyResolve);
    }

    [STAThread]
    static void Main(string[] args)
```

```
    {
      Type t = Type.GetType("ClassLibrary1.Class1", false);
      if (t != null)
      {
        Console.WriteLine(t.Name);
      }
      else
      {
        Console.WriteLine("Тип не найден");
      }


      ClassLibrary2.Class2 c2 = new ClassLibrary2.Class2();
      Console.WriteLine(c2.ToString());
    }

    private static Assembly CurrentDomain_TypeResolve(
                        object sender, ResolveEventArgs args)
    {
      Console.WriteLine("Ищу тип: {0}...", args.Name);

      if (args.Name.Equals("ClassLibrary1.Class1"))
      {
        return Assembly.LoadFrom(
                  @"..\ClassLibrary1\bin\ClassLibrary1.dll");
      }

      return null;
    }

    private static Assembly currentDomain_AssemblyResolve(
                         object sender, ResolveEventArgs args)
    {
      Console.WriteLine("Ищу сборку {0}...", args.Name);
      if (args.Name.IndexOf("ClassLibrary2") >= 0)
      {
        return Assembly.LoadFrom(
                   @"..\ClassLibrary2\bin\ClassLibrary2.dll");
      }
      return null;
    }
  }
}
```

ГЛАВА 5

Отладка и условная компиляция

5.1. Условная компиляция в режиме отладки

Код, заключенный в условие `#ifdef DEBUG`, будет компилироваться только в режиме отладки (debug mode):

```
#if DEBUG
  Console.ReadLine("Режим Debug");
  Console.ReadLine("Для продолжения нажмите клавишу Ввод");
  Console.ReadLine();
#endif
```

С помощью атрибутов можно определить метод класса, который будет компилироваться только в режиме отладки:

```
[System.Diagnostics.Conditional("DEBUG")]
static void DbgMsg()
{
  Console.WriteLine("DEBUG!!!");
}
... ... ... ... ... ... ... ...
// Будет выполняться только в Debug-режиме
DbgMsg();
```

5.2. Методы *ASSERT* и *TRACE*

В C#, так же как и в C++, существует метод `ASSERT`, вызывающий исключение, если некоторое условие не выполняется:

```
System.Diagnostics.Debug.Assert(условие, "Сообщение об ошибке",
                                "Подробности");
```

Метод `Trace` позволяет выводить информацию в окно **Output** отладчика среды:

```
System.Diagnostics.Debug.Trace("Строка информации");
```

В пространстве имен `System.Diagnostics` существует еще один объект: `System.Diagnostics.Trace`. Методы класса `Trace` выполняются, если определено макроимя `TRACE`. Другими словами, методы класса `System.Diagnostics.Debug` выполняются, если определено имя `DEBUG`, а методы класса `System.Diagnostics.Trace` — если определено имя `TRACE`.

5.3. Перенаправление вывода *Trace*

Классы `System.Diagnostics.Debug` и `System.Diagnostics.Trace` имеют свойство `Listeners`, являющееся коллекцией приемников данных, которые посылаются методами `WriteLine`. По умолчанию эта коллекция имеет один приемник, являющийся наследником класса `DefaultTraceListener`. Данные в этом случае посылаются с использованием Win32-функции `OutputDebugString()` и также метода `System.Diagnostics.Debugger.Log()`. Такое поведение удобно при отладке, но при развертывании приложения может потребоваться перенаправление трассировки в файл.

Для перенаправления трассировки в файл используется класс `TextWriterTraceListener`:

```
// Удаляем приемник по умолчанию (если нужно, можно его оставить,
// но его использование занимает существенные ресурсы)
Trace.Listeners.Clear();
// Создаем новый приемник
FileStream fs = new FileStream( @"c:\log.txt", FileMode.Create,
          FileAccess.Write);
Trace.Listeners.Add( new TextWriterTraceListener(fs));

// Эта трассировочная информация будет записана в файл
Trace.WriteLine( @"Эта строка будет записана в log.txt!" );
// Сбрасываем буфер
Trace.Flush();
```

Можно создавать свои наследники приемников, например, для отображения времени выполнения программы:

```
class MyListener : TextWriterTraceListener
{
  public MyListener(Stream s) : base(s)
  {
  }
```

```
  public override void WriteLine(string s)
  {
    Writer.WriteLine("{0:D8} [{1:D4}] {2}",
      Environment.TickCount - m_startTickCount,
      AppDomain.GetCurrentThreadId(),
      s );
  }
  protected int m_startTickCount = Environment.TickCount;
}
```

5.4. Почему дублируются строки в отладочной информации?

Вероятно, были добавлены два одинаковых приемника из разных мест кода. Чтобы избежать такой ситуации перед добавлением приемника, следует проверять его существование.

```
if(Trace.Listeners.Count == 0)
{
  Trace.Listeners.Add(new TextWriterTraceListener(Console.Out));
  Trace.AutoFlush = true;
  Trace.Indent();
}
```

5.5. Трассировка исключения

Следующий код дает информацию об исключении. Его можно использовать в блоке `catch` (листинг 5.1).

Листинг 5.1. Трассировка исключения

```
public static void TraceExceptionCatched(Exception e)
{
  StackTrace st = new StackTrace(1, true);
  StackFrame sf = st.GetFrame(StackTrace.METHODS_TO_SKIP);
  if (sf != null)
  {
    MethodBase mb = sf.GetMethod();
    Trace.WriteLine(e.ToString());
    StringBuilder sb = new StringBuilder();
    sb.Append("catched in File ");
    sb.Append(sf.GetFileName());
    sb.Append(", line ");
```

```
    sb.Append(sf.GetFileLineNumber());
    Trace.WriteLine(sb.ToString());
    sb = new StringBuilder();
    sb.Append("in method ");
    sb.Append(mb.ReflectedType.Name);
    sb.Append(".");
    sb.Append(mb.Name);
    sb.Append("(");
    bool first = true;
    foreach (ParameterInfo pi in mb.GetParameters())
    {
      if (!first)
        sb.Append(", ");
      first = false;
      sb.Append(pi.ParameterType.Name);
      sb.Append(" ");
      sb.Append(pi.Name);
    }
    sb.Append(");");
    Trace.WriteLine(sb.ToString());
  }
}
```

5.6. Трассировка стека вызовов

Для трассировки стека используется класс `System.Diagnostics.StackTrace`:

```
System.Diagnostics.StackTrace s =
         new System.Diagnostics.StackTrace(0);
Console.WriteLine(s.GetFrame(0).GetMethod().Name);
```

Метод `GetFrame(0)` возвращает первый фрейм объекта `StackTrace`, т. е. как раз тот, который в данный момент выполняется. Метод `GetMethod()` возвращает ссылку на объект `MethodBase`, соответствующий методу заданного фрейма.

Соответственно, для получения всего стека вызовов можно использовать код:

```
System.Diagnostics.StackTrace s =
           new System.Diagnostics.StackTrace(0);
for (int i=0; i< s.FrameCount; i++)
{
  Console.WriteLine(s.GetFrame(i).GetMethod().Name);
}
```

Листинг 5.2 демонстрирует пример кода отображения текущего состояния стека вызовов.

Листинг 5.2. Отображение стека вызовов

```
using System;
using System.Diagnostics;
using System.Reflection;
using System.Text;

namespace TraceCallStack
{
  class Class1
  {
    static void Test1()
    {
      Test2();
    }

    static void Test2()
    {
      Console.WriteLine(TraceCallStack());
    }

    public static string TraceCallStack()
    {
      StringBuilder sb = new StringBuilder();

      StackTrace st = new StackTrace(1, true);
      StackFrame sf = st.GetFrame(StackTrace.METHODS_TO_SKIP);
      if (sf != null)
      {
        MethodBase mb = sf.GetMethod();
        sb.Append(mb.ReflectedType.Name);
        sb.Append(".");
        sb.Append(mb.Name);
        sb.Append("(");
        bool first = true;
        foreach (ParameterInfo pi in mb.GetParameters())
        {
          if (!first)
            sb.Append(", ");
          first = false;
          sb.Append(pi.ParameterType.Name);
          sb.Append(" ");
          sb.Append(pi.Name);
        }
```

```
        sb.Append(");");
        int n = StackTrace.METHODS_TO_SKIP+1;
        sf = st.GetFrame(n);
        if (sf != null)
        {
          sb.Append(" called from ");
          do
          {
            mb = sf.GetMethod();
            sb.Append(mb.ReflectedType.Name);
            sb.Append(".");
            sb.Append(mb.Name);
            sb.Append(":");
            sf = st.GetFrame(++n);
          }
          while (sf != null);
        }
      }
      return sb.ToString();
    }

    [STAThread]
    static void Main(string[] args)
    {
      Test1();
      Console.ReadLine();
    }
  }
}
```

5.7. Оценка времени выполнения кода

Оценка времени выполнения кода может осуществляться разными способами, в зависимости от требуемой точности.

5.7.1. Измерение с помощью *TickCount* (наименьшая точность)

Наиболее простой, но не очень точный способ измерения выглядит следующим образом:

```
int start = Environment.TickCount; // начало измерения
// некоторые действия
int end = Environment.TickCount;   // завершение измерения
Console.WriteLine("TickCount=" + (end - start)); // результат
```

5.7.2. Измерение с помощью *Ticks* (средняя точность)

Более точный результат (точность 100 наносекунд) дает метод `Ticks` класса `DateTime`:

```
long startLong = DateTime.Now.Ticks; // начало измерения
// некоторые действия
long endLong = DateTime.Now.Ticks;   // завершение измерения
Console.WriteLine("Ticks=" + (endLong - startLong));
```

5.7.3. Измерение с помощью *QueryPerformance* (высокая точность)

Наиболее точный способ измерения интервалов времени (точность 1/3579545 секунды) предлагает MSDN (статья Q306979):

```
// Импорт функций из Kernel
[System.Runtime.InteropServices.DllImport("kernel32.dll")]
extern static short QueryPerformanceCounter(ref long x);
 [System.Runtime.InteropServices.DllImport("kernel32.dll")]
extern static short QueryPerformanceFrequency(ref long x);

long ctr1 = 0, ctr2 = 0, freq = 0;

// Начало измерения
if (QueryPerformanceCounter(ref ctr1) != 0)
{
  // Некоторые действия

  // Завершение измерения
  QueryPerformanceCounter(ref ctr2);
  // Получение делителя точности
  QueryPerformanceFrequency(ref freq);
  Console.WriteLine("Минимальное разрешение: 1/" + freq + " сек.");
  Console.WriteLine("Время выполнения: " + (ctr2 - ctr1) *
                                    1.0 / freq + " сек.");
}
```

5.8. Как включить полный режим отладки кода?

По умолчанию для отладки включен только безопасный код. В этом случае увидеть и отлаживать вызовы системных модулей нельзя. Для активизации режима отладки всего кода (call stack) следует включить опцию **Unmanaged code debugging**.

5.9. Отладочная информация для ASP.NET

Для включения отладочной информации (в частности, времени выполнения методов страницы) на странице ASP.NET нужно выставить свойство `Trace` в значение `true`:

```
<%@ Page language="c#" Codebehind="test.aspx.cs"
AutoEventWireup="false" Inherits="Receipt.Test"
validateRequest="false" Trace="true"%>
```

5.10. Как отлаживать remoting-код?

Точку останова следует поставить на строку вызова remoting-метода и строку внутри remoting-кода. Запустив код, дождаться срабатывания первой точки останова (до вызова метода), открыть диалог **Processes** в меню **Debug**, выбрать процесс wp_asp.exe, нажать кнопку **Attach** и затем кнопку **Close**. После этого продолжить выполнение программы. В следующий раз останов произойдет внутри remoting-кода.

5.11. Глобальная обработка исключений

Для записи в журнал всех исключений, происходящих в приложении, можно воспользоваться событием `AppDomain.CurrentDomain.UnhandledException` (листинг 5.3). *См. также разд. 22.6 и 24.12.*

Листинг 5.3. Обработка `AppDomain.CurrentDomain.UnhandledException`

```
using System;

namespace UnhandledExceptionTest
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Обработчик исключений
      AppDomain.CurrentDomain.UnhandledException +=
                    new UnhandledExceptionEventHandler(
                                  CurrentDomain_UnhandledException);

    // Генерируем исключение "деление на ноль"
      for (int i=0; i<10; i++)
```

```
      {
        int j = 10/(i-9);
      }
    }

    // Обрабатываем все упущенные исключения
    private static void CurrentDomain_UnhandledException(
                  object sender, UnhandledExceptionEventArgs e)
    {
      Console.WriteLine("CurrentDomain_UnhandledException:");
      Console.WriteLine(sender.ToString());
      Console.WriteLine(e.ExceptionObject.ToString());
      Console.WriteLine(e.IsTerminating);
    }
  }
}
```

ГЛАВА 6

Командная строка, свойства программы, конфигурационные файлы

6.1. Параметры командной строки

Параметры командной строки передаются с помощью параметра `args`, передаваемого в метод `Main`. В командной строке каждый параметр разделяется пробелом. Символы, заключенные в кавычки, являются одним параметром, независимо от пробелов внутри кавычек. Например, командная строка <1 2 3> будет передавать три параметра, а строка <"1 2" 3> будет передавать два параметра, а именно первый параметр <1 2> и второй параметр <3>. Листинг 6.1 содержит пример работы с командной строкой.

Листинг 6.1. Вывод параметров командной строки

```
using System;
namespace Args
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      foreach (string arg in args)
        Console.WriteLine(arg);
    }
  }
}
```

6.2. Имя исполняемого файла

В C++ нулевым параметром командной строки передавалось полное имя исполняемого модуля (т. е. имя файла), в C# нулевой параметр передает первый параметр командной строки. Другими словами, массив `args` содержит только параметры командной строки.

Для получения полного пути исполняемого файла консольной программы предназначен объект `System.Environment.CommandLine` (листинг 6.2). Для приложений типа Windows Forms можно использовать свойство `Application.ExecutablePath`. Для получения имени сборки применяется вызов:

```
System.Reflection.Assembly.GetEntryAssembly().GetName().Name
```

Листинг 6.2. Имя исполняемого файла

```
using System;

namespace CommandLine
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      Console.WriteLine(System.Environment.CommandLine.ToString());
    }
  }
}
```

6.3. Установка свойства *CompanyName*

Установка свойства `CompanyName` (его можно также получить через вызов `System.Windows.Forms.Application.CompanyName`) производится с помощью соответствующего свойства в файле AssemblyInfo.cs:

```
[assembly: AssemblyCompany("MyCompany Inc.")]
```

Там же можно установить и другие атрибуты программы, например, версию модуля.

6.4. Конфигурационный файл

Каждая программа может иметь конфигурационный файл, сохраняющий настройки программы.

Для Web-приложений этот файл называется web.config.

Для консольных и программ Windows Form этот файл называется app.config. Этот файл должен быть включен в проект. При компиляции он копируется в каталог исполняемых файлов с именем *имя_модуля*.config. Например, если компилируется программа test.exe, имя файла конфигурации будет test.config.

Обычный вид конфигурационного файла:

```
<?xml version="1.0" standalone="yes" ?>
<configuration>
  <appSettings>
    <add key="key1" value="val1" />
    <add key="key2" value="val2" />
  </appSettings>
</configuration>
```

Для доступа к значениям параметров используется класс `ConfigurationSettings`:

```
Console.WriteLine(ConfigurationSettings.AppSettings["key1"]);
```

О создании нестандартного конфигурационного файла *см. разд. 6.6*.

6.5. Где найти класс *ConfigurationManager*?

В C# 2.0 при использовании метода `ConfigurationSettings.GetConfig()` компилятор выдает сообщение, что этот метод устарел, и следует использовать метод `System.Configuration.ConfigurationManager.GetSection()`. Однако такого класса в пространстве имен `System.Configuration` нет. Проблема решается добавлением ссылки (reference) на пространство имен `System.Configuration` (сборка `System.Configuration.dll`).

6.6. Нестандартный конфигурационный файл

Удобство стандартного конфигурационного файла в том, что его имя совпадает с именем исполняемого файла. Таким образом, нет необходимости сохранять имя конфигурации при использовании модулей в разных исполняемых файлах. Модуль может использоваться как в консольной программе, так и в Web-приложении, но конфигурация всегда будет доступна через класс `ConfigurationSettings`.

Однако очень часто формата стандартных настроек не хватает. Для расширения функциональности можно описывать пользовательские обработчики блоков конфигурации. Описание выглядит следующим образом:

```
<configSections>
  <sectionGroup name="имя группы">
```

```
    <section name="имя секции" type="тип обработчика" />
  </sectionGroup>
</configSections>
```

Обработчики секций могут быть как стандартными, так и определяемыми пользователем. Определяемые пользователем обработчики должны находиться в dll-модуле. Пример, приведенный далее, показывает четыре способа обработки секций: стандартным обработчиком, чтением конфигурации как XML-узла, с помощью пользовательского объекта и с помощью сериализации (листинги 6.3—6.9).

Листинг 6.3. Пример конфигурационного файла

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

 <configSections>
  <sectionGroup name="variableConfig">
   <section name="variableSection"
        type="System.Configuration.NameValueSectionHandler" />
  </sectionGroup>
  <sectionGroup name="taskListConfig">
   <section name="taskConfiguration"
        type="ConfigHandler.CustomSectionHandler, ConfigHandler" />
  </sectionGroup>
  <sectionGroup name="addressConfig">
   <section name="addressConfiguration"
        type="ConfigHandler.AddressSectionHandler, ConfigHandler" />
  </sectionGroup>
  <sectionGroup name="addressXMLConfig">
   <section name="addressXMLConfiguration"
         type="ConfigHandler.XMLSectionHandler, ConfigHandler" />
  </sectionGroup>
 </configSections>

 <variableConfig>
  <variableSection>
   <add key="v1" value="val1"/>
   <add key="v2" value="val2"/>
  </variableSection>
 </variableConfig>

 <taskListConfig>
  <taskConfiguration>
```

```
   <task value="start_1" time="10" />
   <task value="start_2" time="20" />
   <task value="start_3" time="30" />
  </taskConfiguration>
 </taskListConfig>

 <addressConfig>
  <addressConfiguration>
    <company>PVASoft</company>
    <country>Russia</country>
  </addressConfiguration>
 </addressConfig>

 <addressXMLConfig>
  <addressXMLConfiguration>
    <company>PVASoft1</company>
    <country>Russia1</country>
  </addressXMLConfiguration>
 </addressXMLConfig>

</configuration>
```

Листинг 6.4. Обработчики секций конфигурации (отдельная сборка)

```
using System;
using System.Configuration;
using System.Xml;

namespace ConfigHandler
{
  // Обработчик конфигурации возвращает XmlElement
  public class CustomSectionHandler : IConfigurationSectionHandler
  {
    public object Create(object parent, object
                         configContext, XmlNode section)
    {
      XmlElement root = (XmlElement)section;
      return root;
    }
  }

  public class Address
  {
    private string company;
    public string Company
```

```
    {
      get { return company; }
    }

    private string country;
    public string Country
    {
      get { return country; }
    }

    public Address(string company, string country)
    {
      this.company = company;
      this.country = country;
    }
  }

  // Обработчик конфигурации возвращает значения
  public class AddressSectionHandler : IConfigurationSectionHandler
  {
    public object Create(object parent,
                  object configContext, XmlNode section)
    {
      string company =
                section.SelectSingleNode("company").InnerText;
      string country =
                section.SelectSingleNode("country").InnerText;
      return new Address(company, country);
    }
  }
}
```

Листинг 6.5. Использование обработчиков секций конфигурации

```
using System;
using System.Collections.Specialized;
using System.Configuration;
using System.Xml;

using ConfigHandler;

namespace TestApplication
{
  class Class1
```

```
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Обычное получение параметров
      Console.WriteLine(ConfigurationSettings.AppSettings["key1"]);

      // Стандартный обработчик дополнительной секции
      // <add key="v1" value="val1"/>
      NameValueCollection vars = (NameValueCollection)
               ConfigurationSettings.GetConfig(
                      "variableConfig/variableSection");
      Console.WriteLine(vars["v1"]);

      // Чтение XML-узла
      XmlElement cfg =
               (XmlElement)ConfigurationSettings.GetConfig(
                      "taskListConfig/taskConfiguration");
      foreach (XmlNode node in cfg.ChildNodes)
      {
        if (node.NodeType == XmlNodeType.Element)
          Console.WriteLine(node.OuterXml);
      }

      // Чтение специальным классом
      Address adr =(Address)
               ConfigurationSettings.GetConfig(
                    "addressConfig/addressConfiguration");
      Console.WriteLine("company={0}, country={1}",
                           adr.Company, adr.Country);
    }
  }
}
```

Листинг 6.6. Описание стандартного класса конфигурации

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <configSections>
 <section name="TestLoaderConfiguration"
         type="System.Configuration.DictionarySectionHandler" />
 </configSections>
```

```
 <TestLoaderConfiguration>
 </TestLoaderConfiguration>
</configuration>
```

Листинг 6.7. Использование стандартного класса конфигурации

```
private IDictionary  loaderSectionHandler;
loaderSectionHandler = (IDictionary)
        ConfigurationSettings.GetConfig("TestLoaderConfiguration");
```

Листинг 6.8. Чтение секций конфигурации с помощью сериализации

```
using System;
using System.Configuration;
using System.Xml;
using System.Xml.Serialization;

namespace ConfigHandler
{
  // Обработчик конфигурации возвращает XmlElement
  public class XMLSectionHandler : IConfigurationSectionHandler
  {
    public object Create(object parent, object configContext,
                         XmlNode section)
    {
      XmlSerializer ser = new XmlSerializer(typeof(XMLAddress));
      XmlNodeReader nr = new XmlNodeReader(section);
      try
      {
        XMLAddress address = ser.Deserialize(nr) as XMLAddress;
        return address;
      }
      finally
      {
        nr.Close();
      }
    }
  }

  [XmlRoot(ElementName="addressXMLConfiguration")]
  public class XMLAddress
  {
```

```
    public XMLAddress()
    {
    }

    private string company;
    [XmlElement(ElementName="company")]
    public string Company
    {
      get { return company;  }
      set { company = value; }
    }

    private string country;
    [XmlElement(ElementName="country")]
    public string Country
    {
      get { return country; }
      set { country = value; }
    }
  }
}
```

Листинг 6.9. Использование класса `XMLSectionHandler`

```
// Чтение через XML
XMLAddress adrXML =(XMLAddress) ConfigurationSettings.GetConfig(
                    "addressXMLConfig/addressXMLConfiguration");
Console.WriteLine("company={0}, country={1}",
                      adrXML.Company, adrXML.Country);
```

6.7. Как заставить программу использовать только Framework 2.0?

Для того чтобы заставить программу использовать только Framework 2.0, нужно в конфигурационном файле программы написать (см. RSDN FAQ):

```
<configuration>
  <startup>
    <!-- Тут - номер нужной версии -->
    <supportedRuntime version="v2.0.40607" />
  </startup>
</configuration>
```

6.8. Версия и информация о модуле

Для указания версии и стандартной информации о модуле в проект должен быть включен файл AssemblyInfo.cs, содержащий соответствующие строки:

```
[assembly: AssemblyTitle("")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
[assembly: AssemblyVersion("1.2.3.4")]
```

Для получения этой информации используется класс `System.Reflection.Assembly`:

```
System.Reflection.Assembly assembly =
               System.Reflection.Assembly.GetCallingAssembly();
Console.WriteLine(assembly.GetName().Version.ToString());
```

Для приложений Windows Form используется вызов:

```
string strVersion = Application.ProductVersion;
```

Часть информации доступна с помощью полей:

- `System.Environment.Version.Build` — номер сборки;
- `System.Environment.Version.Major` — старший номер версии;
- `System.Environment.Version.Minor` — младший номер версии;
- `System.Environment.Version.Revision` — номер ревизии.

6.9. Как обновлять *AssemblyInfo* для всего проекта сразу?

Удобнее всего сделать один файл AssemblyInfo.cs и подключить его к каждому из проектов как ссылку. Для этого для каждого проекта нужно через меню **Add Existing Item** выбрать файл AssemblyInfo.cs, но добавить его не просто нажатием кнопки **Open**, а, открыв меню, связанное с этой кнопкой, выбрать пункт **Link File**. В проект будет добавлена только ссылка на файл.

ГЛАВА 7

Преобразования

7.1. Преобразование числа в шестнадцатеричную строку

Преобразование числа в шестнадцатеричную строку производится с помощью форматной строки:

```
int i=55;
Console.WriteLine("0x{0:X}", i);
```

Либо с помощью метода `ToString()` самого класса:

```
string HexStr = i.ToString("X");
```

Количество знаков в полученной строке можно задавать после символа формата (строка будет дополнена нулями слева). Например, преобразование числа в строку из 4-х символов будет выглядеть так:

```
string HexStr = i.ToString("X4");
```

7.2. Преобразование строки в число

Метод `Parse` позволяет преобразовать строку в число:

```
string DecStr="31";
int j= int.Parse(DecStr);
```

Еще один способ — использование методов класса `Convert`:

```
int j = Convert.ToInt32(DecStr);
```

В данном случае, разницы между ними почти нет, за исключением случая `null`: `Convert.ToInt32(null)` вернет ноль, а `Int32.Parse(null)` вызовет исключение `ArgumentNullException`.

7.3. Преобразование строки с пробелами в число

Специальные флаги позволяют преобразовать строку, содержащую пробелы, в число:

```
// Разрешаем разделители в начале
i = int.Parse(" 123", NumberStyles.AllowLeadingWhite, null);
Console.WriteLine(i);

// Разрешаем разделители и символы знака
i = int.Parse(" -123 ", NumberStyles.Integer, null);
Console.WriteLine(i);
```

7.4. Преобразование шестнадцатеричной строки в число

Метод `Parse` позволяет преобразовать шестнадцатеричную строку без префикса в число:

```
string HexStr="33A";
int j =
   int.Parse(HexStr, System.Globalization.NumberStyles.HexNumber);
```

Еще один способ — использование методов класса `Convert`:

```
int j = Convert.ToInt32(HexStr, 16);
```

7.5. Преобразование двоичной строки в число

Методы класса `Convert` позволяют преобразовать двоичную строку в число:

```
string BinStr="1011";
int b = Convert.ToInt32(BinStr, 2);
```

7.6. Преобразование числа в двоичную строку

Метод `Convert.ToString(myValue, 2)` преобразует число в двоичную строку, например, `Convert.ToString(128, 2)` вернет `"10000000"`.

7.7. Преобразование числа с плавающей точкой в строку

Преобразование числа с плавающей точкой в строку производится с помощью метода `string.Format()` или `docuble.ToString()`(листинг 7.1).

Листинг 7.1. Преобразование числа с плавающей точкой в строку

```
using System;
namespace Float2String
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      double d = 3.1415926;
      Console.WriteLine("{0}", d);                  // 3,1415926
      Console.WriteLine("{0:F2}", d);               // 3,14
      Console.WriteLine(string.Format("{0}", d));   // 3,1415926
      Console.WriteLine(string.Format("{0:F3}", d));// 3,142
      Console.WriteLine(d.ToString("F2"));          // 3,14
      Console.WriteLine(d.ToString("E"));           // 3,141593E+000
      Console.WriteLine(d.ToString("C"));           // 3,14р.
      Console.WriteLine(d.ToString("G"));           // 3,1415926
      Console.WriteLine(d.ToString("R"));           // 3,1415926
    }
  }
}
```

7.8. Преобразование строки в число с плавающей точкой

Для преобразования строки в число типа `double` можно также использовать два метода:

```
Convert.ToDouble(str);
```

или

```
Double.Parse(str);
```

Но, в отличие от `int`, класс `double` предоставляет еще один удобный метод. Метод `Double.TryParse(...)` возвращает `true`, если преобразование прошло успешно. Пример конвертирования показан в листинге 7.2. В последнем примере и точка, и запятая заменяются на текущее значение регионального разделителя, что позволяет конвертировать строку, независимо от используемого разделителя.

Листинг 7.2. Преобразование строки в число с плавающей точкой

```
using System;
using System.Globalization;
```

```
namespace DoubleConvert
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      string str_1 = "55,6";
      Console.WriteLine(Convert.ToDouble(str_1));

      // Вызовет исключение, если десятичный
      // разделитель — запятая, а не точка
      string str_2 = "55.6";
      try
      {
        Console.WriteLine(Convert.ToDouble(str_2));
      }
      catch (Exception e)
      {
        Console.WriteLine(e.Message);
      }

      // TryParse умеет понимать не только десятичные
      // разделители, но и разделители разрядов
      double retNum;
      Double.TryParse(str_1, NumberStyles.Any,
             NumberFormatInfo.InvariantInfo, out retNum);
      Console.WriteLine(retNum); // преобразуется в 556,
                                 // если запятая является
                                 // разделителем разрядов
      Double.TryParse(str_2, NumberStyles.Any,
             NumberFormatInfo.InvariantInfo, out retNum);
      Console.WriteLine(retNum);   // преобразуется в 55.6,
                                   // если точка является
                                   // десятичным разделителем
      Console.WriteLine(Double.Parse(str_1));

      // Заменяем и точку, и запятую на текущее значение
      // десятичного разделителя
      string str_3 = "55,6";
      string decimal_sep =
           NumberFormatInfo.CurrentInfo.CurrencyDecimalSeparator;
      str_3 = str_3.Replace(".", decimal_sep);
      str_3 = str_3.Replace(",", decimal_sep);
```

```
      Console.WriteLine(decimal.Parse(str_3));
    }
  }
}
```

7.9. Перекодировка текста

Для перекодировки текста можно использовать метод класса пространства имен `Text`. Исходную строку следует сначала перевести в байтовый массив, а затем в нужную кодировку, например:

```
string SrcStr = ...;
byte [] srcbytes = System.Text.ASCIIEncoding.ASCII.GetBytes(SrcStr);
System.Text.UTF8Encoding utf = new System.Text.UTF8Encoding();
string UtfStr = utf.GetString(srcbytes);
```

7.10. Преобразование в *Base64* и обратно

Для преобразования строки в `Base64` можно использовать метод `ToBase64String()` класса `Convert`:

```
string SrcStr = "Пример текста";
byte [] srcbytes = System.Text.UTF8Encoding.UTF8.GetBytes(SrcStr);
string base64 = Convert.ToBase64String(srcbytes);
```

Для обратного преобразования предназначен метод `FromBase64String()`.

7.11. Преобразование из Win1251 в KOI8 и обратно

Класс `Encoding` применяется для преобразования кодировок текста (листинг 7.3).

Листинг 7.3. Преобразование кодировок текста

```
public static string Koi8rToWin1251(string source)
{
  Encoding win1251 = Encoding.GetEncoding("windows-1251");
  byte[] srcBytes = win1251.GetBytes(source);
  byte[] dstBytes = Encoding.Convert(
          Encoding.GetEncoding("koi8-r"), win1251, srcBytes);
  return win1251.GetString(dstBytes);
 }
```

```
public static string Win1251ToKoi8r(string source)
{
 Encoding koi8r = Encoding.GetEncoding("koi8-r");
 byte[] srcBytes = koi8r.GetBytes(source);
 byte[] dstBytes = Encoding.Convert(
         Encoding.GetEncoding("windows-1251"), koi8r, srcBytes);
 return koi8r.GetString(dstBytes);
}
```

7.12. Преобразование цвета в строку и обратно

Для преобразования цвета в строку можно использовать либо методы класса `ComponentModel.TypeDescriptor`, либо методы класса `Drawing.ColorConverter` (листинг 7.4).

Листинг 7.4. Преобразование цвета в строку и обратно

```
// Преобразование из строки в цвет
System.Drawing.Color redColor = (System.Drawing.Color)
  System.ComponentModel.TypeDescriptor.GetConverter(
    typeof(System.Drawing.Color)).ConvertFromString("Red");
// Преобразование из цвета в строку
string redString =
  System.ComponentModel.TypeDescriptor.GetConverter(
    typeof(System.Drawing.Color)).ConvertToString(
    System.Drawing.Color.Red
  );
// Использование класса ColorConverter
System.Drawing.ColorConverter ccv =
          new System.Drawing.ColorConverter();
this.BackColor =
    (System.Drawing.Color) ccv.ConvertFromString("Red");
```

7.13. Преобразование цвета в HTML-формат

Класс `System.Drawing.Imaging.ColorTranslator` позволяет преобразовать цвет в HTML-формат и обратно (листинг 7.5).

Листинг 7.5. Преобразование цвета в HTML-формат

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
```

```
namespace ColorTranslatorTest
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      string html = ColorTranslator.ToHtml(Color.Red);
      Color cl = ColorTranslator.FromHtml(html);
      Console.WriteLine("{0}={1}", cl.Name, html);
    }
  }
}
```

7.14. Преобразование цвета в целое число и обратно

Для преобразования цвета в целое число и обратно можно использовать методы `FromArgb` и `ToArgb`:

```
int blueInt = Color.Blue.ToArgb();
Color newColor = Color.FromArgb(blueInt);
```

7.15. Преобразование HTML-текста

Очень часто требуется заменить в строке недопустимые HTML-символы, заменив их соответствующими аббревиатурами. Для этого применяется метод `System.Web.HttpUtility.HtmlEncode`. Обратное преобразование производит метод `HttpUtility.HtmlDecode`. Для использования этих методов в консольных (листинг 7.6) или WinForm-приложениях требуется вручную подключить библиотеку `System.Web`.

Листинг 7.6. Преобразование HTML-текста

```
using System;
using System.Web;

namespace HTMLTextConverter
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
```

```
    {
      string source = "test&test;";
      string html = HttpUtility.HtmlEncode(source);
      Console.WriteLine(html);
      string text = HttpUtility.HtmlDecode(html);
      Console.WriteLine(text);
    }
  }
}
```

7.16. Преобразование массива байтов в базовые типы и обратно

Преобразовать массив байтов в базовые типы и обратно можно с помощью класса `System.BitConverter`, например:

```
// Вернет шестнадцатеричную строку "16-22-37-00"
BitConverter.ToString (new byte[] {22,34,55,0}, 0)
// Вернет число 5,06318E-39
BitConverter.ToSingle (new byte[] {22,34,55,0}, 0)
// Вернет True
BitConverter.ToBoolean(new byte[] {22,34,55,0}, 0)
// Вернет False
BitConverter.ToBoolean(new byte[] {0 , 0, 0,0}, 0)
```

Обратное преобразование выполняется с помощью метода `GetBytes`:

```
byte [] arr = BitConverter.GetBytes(77.5);
foreach (byte b in arr)
Console.Write("{0} ", b);
Console.WriteLine();
```

ГЛАВА 8

Массивы, списки, таблицы, перечисления

Пример перечисления:

```
enum Button
{
  Start,
  Stop ,
  Play ,
  Next ,
  Prev
}
```

8.1. Отображение всех элементов перечисления

Перечисление всех элементов можно осуществить следующим образом:

```
Button [] Buttons = (Button[]) Enum.GetValues(typeof(Button));
foreach (Button btn in Buttons)
{
  // Показывает числовое значение и имя
  Console.WriteLine(btn.ToString("D") + " = " + btn.ToString("G"));
}
```

Еще один пример — получение всех цветов, зарегистрированных в системе:

```
string[] nm= Enum.GetNames(typeof(System.Drawing.KnownColor));
```

8.2. Перевод перечисления в строковый вид и обратно

Метод `ToString()` позволяет получить строковое представление элемента перечисления:

```
Button b = Button.Stop;
// Отобразит "Stop"
Console.WriteLine(b.ToString());
// Отобразит "Stop"
Console.WriteLine(b.ToString("G"));
// Отобразит "1"
Console.WriteLine(b.ToString("D"));
```

Или без переменной:

```
// Отобразит "Next"
Console.WriteLine(Button.Next.ToString());
```

Обратное преобразование:

```
// Отобразит "Prev"
Console.WriteLine(Enum.GetName(typeof(Button), 4).ToString());
// Отобразит "Stop"
Console.WriteLine(Enum.GetName(typeof(Button), b).ToString());
```

Преобразование из строки:

```
// Отобразит "Prev"
Button c =(Button) Enum.Parse(typeof(Button),"Prev");
Console.WriteLine(c.ToString());
```

8.3. Проверка наличия элемента в перечислении

Проверку наличия элемента в перечислении можно выполнить с помощью метода `IsDefined`:

```
if (Enum.IsDefined(typeof(Button), "Next"))
  Console.WriteLine("Есть элемент с таким именем");
if (Enum.IsDefined(typeof(Button), 3))
  Console.WriteLine("Есть такой элемент с номером 3");
```

8.4. Одномерные массивы постоянного размера

Создание массива производится с помощью оператора `new`:

```
int  [] myArray;           // описание массива
myArray = new int[50];     // создание массива
```

Базовым типом массива может быть любой тип:

```
Button  [] myArray;        // описание массива
myArray = new Button [10]; // создание массива
```

Описание и инициализацию массива можно совместить:

```
Button  [] myArray = new Button [10];
```

Доступ к элементам массива производится с помощью квадратных скобок. Элементы массива нумеруются с нуля:

```
myArray[0] = 10;
myArray[1] = 20;
```

8.4.1. Изменение размерности массива

Изменение размерности массива производится с помощью метода `CreateInstance` (см. [2], с. 276). Вообще говоря, осуществляется создание нового массива и копирование в него элементов старого:

```
public static Array Redim(Array origArray,Int32 desiredSize)
{
  // Определяем тип элемента массива
  Type t =origArray.GetType().GetElementType();
  // Создаем новый массив с требуемым числом элементов.
  // Тип массива должен совпадать с типом исходного массива
  Array newArray = Array.CreateInstance(t,desiredSize);
  // Копируем элементы из исходного массива в новый массив
  Array.Copy(origArray,0,
    newArray,0,Math.Min(origArray.Length,desiredSize));
  // Возвращаем новый массив
  return newArray;
}
```

Использование этого метода выглядит следующим образом:

```
int [] myArray = new int[5];
Console.WriteLine(myArray.Length.ToString());
myArray = (int[]) Redim(myArray, 10);
Console.WriteLine(myArray.Length.ToString());
```

Однако в таких случаях все-таки лучше использовать массивы переменной длины (класс `ArrayList`), т. к. производительность операции измерения размерности обычного массива весьма низкая.

8.4.2. Перебор всех элементов массива

Для массивов можно использовать как обычный оператор `for`:

```
for (int index=0; index< myArray.Length; index++)
{
  // использование myArray[index]
}
```

так и оператор `foreach`:

```
foreach (int value in myArray)
{
  // использование value
}
```

Однако второй способ не позволяет изменять значение элементов массива и применяется только для перебора (доступ только для чтения).

Следует отметить также, что первая конструкция выполняется быстрее, чем `foreach` или конструкция вида

```
int len = myArray.Length;
for (int index=0; index< len; index++)
```

Дело в том, что первая конструкция распознается как специальный паттерн (шаблон), и компилятором создает код, специально оптимизированный для перебора всех элементов массива (проще говоря, проверка на границы массива выносится за пределы цикла, т. к. компилятор видит явное ограничение счетчика цикла).

8.4.3. Изменение порядка элементов на обратный

Метод `Reverse` класса `Array` позволяет изменить порядок элементов массива на обратный:

```
Array.Reverse(myArray);
```

Этот метод имеет еще одну реализацию, позволяющую изменять порядок элементов частично:

```
Array.Reverse(myArray, 0, 5);
```

8.4.4. Поиск

Для поиска элемента в одномерном массиве можно использовать методы класса `Array`:

- `Array.IndexOf(Array` *`array`*`, object` *`value`*`)` — поиск первого вхождения элемента в массив;
- `Array.IndexOf(Array` *`array`*`, object` *`value`*`, int` *`startindex`*`)` — поиск первого вхождения элемента в массив, начиная с указанного индекса;
- `Array.IndexOf(Array` *`array`*`, object` *`value`*`, int` *`startindex`*`, int` *`count`*`)` — поиск первого вхождения элемента в массив в указанном интервале индексов.

Аналогично, метод `Array.LastIndexOf` ищет последнее вхождение элемента в массиве.

Для отсортированных массивов можно использовать метод `Array.BinarySearch`. При этом тип элемента массива должен иметь реализацию интерфейса `IComparer`.

8.4.5. Копирование элементов массива

Скопировать элементы из одного массива в другой можно с помощью метода:

```
System.Buffer.BlockCopy(
  sourceArray,       // Источник
  sourceOffset,      // Смещение в источнике
  destinationArray,  // Куда копировать
  destinationOffset, // Смещение в приемнике
  count              // Сколько копировать
);
```

8.4.6. Преобразование массива одного типа в массив другого типа

Преобразовать массив к другому типу можно с помощью метода `CopyTo()`:

```
int [] iarray = new int[10];      // Исходный массив
for (int i=0; i< iarray.Length; i++)
  iarray[i] = i;

double[] darray = new double[10]; // Новый массив
iarray.CopyTo(darray, 0);
```

Однако, если элементы массивов не совместимы, будет вызвано исключение. Например, следующий код не работает:

```
string[] sarray = new string[10];
iarray.CopyTo(sarray, 0); // Исключение!
```

8.5. Многомерные массивы постоянного размера

Описание и создание многомерного массива выглядит следующим образом:

```
string [,] myArray2 = new string[5, 10];     // двумерный массив
string [,,] myArray3 = new string[5, 10, 2]; // трехмерный массив
```

Важно отметить, что описание многомерного массива производится именно с помощью запятых в квадратных скобках, а не несколькими квадратными скобками, как с C++.

8.6. Невыровненный массив

Язык C# поддерживает невыровненные массивы (jagged array, другое название — вложенные). В отличие от многомерных массивов, вложенные массивы могут иметь разное количество элементов для каждой размерности. Описание такого массива выглядит как описание многомерного массива с C++:

```
int [][] myArray = new int[5][]; // первая размерность 5
myArray[0] = new int [10];       // вторая размерность 10
myArray[1] = new int [ 4];       // вторая размерность 4
myArray[2] = new int [15];       // вторая размерность 15
```

8.7. Свойства массивов

В .NET массивы являются наследниками класса `System.Array`, который кроме самих элементов массива хранит некоторую дополнительную информацию (листинги 8.1 и 8.2):

- `Rank` — возвращает размерность массива;
- `Length` — возвращает общее число элементов массива;
- `GetLength(int dimension)` — возвращает число элементов в указанном измерении массива;
- `GetLowerBound(int dimension)` — возвращает нижнюю границу указанной размерности массива;
- `GetUpperBound(int dimension)` — возвращает верхнюю границу указанной размерности массива.

Следует учитывать, что размерности многомерных и невыровненных массивов — не одно и то же.

Листинг 8.1. Свойства многомерного массива

```
int [,] myArray = new int[3,2];
myArray[0, 0] = 100;
myArray[0, 1] = 101;
myArray[1, 0] = 200;
myArray[1, 1] = 201;
myArray[2, 0] = 300;
myArray[2, 1] = 301;

// Напечатает 2
Console.WriteLine("Rank={0}",myArray.Rank);
```

```
// Напечатает 6 (т. е. 3 x 2)
Console.WriteLine("Length={0}",myArray.Length);
// Напечатает 3 (размер первого измерения)
Console.WriteLine("GetLength(0)={0}",myArray.GetLength(0));
// Напечатает 3 (размер второго измерения)
Console.WriteLine("GetLength(1)={0}",myArray.GetLength(1));
```

Листинг 8.2. Свойства невыровненного массива

```
int [][] myArray = new int[3][];
myArray[0] = new int[1];
myArray[1] = new int[2];
myArray[2] = new int[3];

myArray[0][0] = 100;

myArray[1][0] = 200;
myArray[1][1] = 201;

myArray[2][0] = 300;
myArray[2][1] = 301;
myArray[2][2] = 302;

// Напечатает 1
Console.WriteLine("Rank={0}",myArray.Rank);
// Напечатает 3
Console.WriteLine("Length={0}",myArray.Length);
// Напечатает 1
Console.WriteLine("GetLength(0)={0}",myArray[0].GetLength(0));
// Напечатает 2
Console.WriteLine("GetLength(1)={0}",myArray[1].GetLength(0));
```

8.8. Методы, возвращающие массивы

Рихтер рекомендует возвращать массивы нулевой длины, а не `null` (см. [2], с. 270). Действительно, следующий код будет выполняться корректно, даже если в массиве нет элементов:

```
// Пример простого и понятного кода
Appointment [] appointments = GetAppointmentsForToday();
for (Int32 a =0,n =appointments.Length; a <n; a++){
  ...
}
```

Необходимость проверки на null делает код более тяжеловесным:

```
// Пример более сложного кода
Appointment [] appointments = GetAppointmentsForToday();
if (appointments !=null){
  for (Int32 a =0,n = appointments.Length; a <n; a++){
    ...
  }
}
```

8.9. Массивы переменного размера

Для создания массива переменной длины используется класс System.Collection.ArrayList:

```
ArrayList myList = new ArrayList();
```

Добавление элементов производится с помощью метода Add:

```
myList.Add("Первый элемент");
myList.Add(2);
myList.Add(3.1415);
```

Массивы переменной длины имеют те же свойства, что и обычные массивы, например, можно использовать оператор foreach для обхода всех элементов:

```
foreach (object obj in myList)
{
  Console.WriteLine(obj.ToString());
}
```

8.10. Массив битов

Класс System.Collection.BitArray позволяет создавать битовые массивы. Экземпляры этого класса нельзя создавать с помощью оператора new, а только из массива или другого экземпляра BitArray.

```
// Создаем массив битов из обычного массива
bool[] bits = {true, false, false, true, true};
System.Collections.BitArray Bits =
                     new System.Collections.BitArray(bits);
// Создаем массив битов из byte-массива.
// Будет добавлено 8 x 3 = 24 элемента
byte[] bytes = {55, 33, 1};
System.Collections.BitArray Bits =
                    new System.Collections.BitArray(bytes);
```

```
// Создаем массив битов из int-массива.
// Будет добавлено 32 x 3 = 96 элементов
int[] ints = {55, 33, 1};
System.Collections.BitArray Bits =
                    new System.Collections.BitArray(ints);
```

При создании битового массива сначала добавляются младшие биты. Следующий код создаст массив битов `{1,0,0,0,0,0,0,0}`:

```
byte[] bytes = {1};
System.Collections.BitArray Bits =
                    new System.Collections.BitArray(bytes);
```

Битовый массив имеет следующие методы:

- метод `Not` инвертирует состояние всех битов массива;
- метод `Set(int` *`index`*`, bool` *`value`*`)` устанавливает выбранный элемент в указанное значение;
- метод `SetAll(bool` *`value`*`)` устанавливает все элементы массива в указанное значение.

Между битовыми массивами одной размерности можно производить операции `And`, `Or` и `Xor`. Например:

```
bool[] bits1 = {true, false, false, true, true};
System.Collections.BitArray Bits1 =
                    new System.Collections.BitArray(bits1);
bool[] bits2 = {false, false, false, true, false};
System.Collections.BitArray Bits2 =
                    new System.Collections.BitArray(bits2);
Bits1.And(Bits2);
```

8.11. Хранение таблицы "ключ/значение"

Класс `System.Collections.Hashtable` предназначен для хранения пар "ключ/значение". Особенность этого класса в том, что для хранения данных применяется специальная *хэш-таблица*. Эта таблица содержит *хэш-коды* объектов, представляющих собой уникальное значение, соответствующее содержимому объекта. Хэш-таблица позволяет оптимизировать и поддерживать на постоянном уровне операции поиска, считывания и записи данных даже для больших объемов информации.

Класс `Hashtable` имеет следующие основные методы:

- `int Count` — свойство, доступное только для чтения, возвращающее число элементов таблицы;
- `void Add(object` *`key`*`, object` *`value`*`)` — добавляет пару "ключ/значение";

- `void Clear()` — очищает таблицу;
- `bool ContainsKey(object` *`key`*`)` — возвращает `true`, если ключ уже содержится в таблице;
- `bool ContainsValue(object` *`value`*`)` — возвращает `true`, если значение уже содержится в таблице;
- `void Remove(object` *`key`*`)` — удаляет пару с заданным ключом.

Доступ к элементам таблицы осуществляется так же, как и к элементам массива — с помощью квадратных скобок, только вместо индекса массива указывается ключ (можно сказать, что ключ является индексом хэш-таблицы).

8.11.1. Добавление элементов

Предположим, что мы храним набор уникальных строковых ключей и соответствующий ключу некоторый числовой счетчик. При добавлении нового элемента мы устанавливаем счетчик в 1, а при попытке добавления уже существующего элемента — просто увеличиваем счетчик на 1. Добавление нового элемента в этом случае может выглядеть следующим образом:

```
string key = ...;
if (dict.ContainsKey(key))        // Элемент уже существует?
{
  dict[key]= ((int) dict[key]+1); // Если да, увеличить счетчик
}
else
{
  dict.Add(key, 1); // Если нет, то добавить элемент со счетчиком 1
}
```

8.11.2. Отображение содержимого *Hashtable*

Элементы `System.Collections.Hashtable` являются экземплярами класса `System.Collections.DictionaryEntry`, который можно использовать для доступа к элементам словаря с помощью оператора `foreach`:

```
foreach (DictionaryEntry DE in dict)
{
   Console.WriteLine(DE.Key+"    "+DE.Value);
}
```

8.11.3. Порядок элементов *Hashtable*

Класс `System.Collections.Hashtable` не гарантирует сохранения порядка элементов, т. е. порядок добавления может не совпадать с порядком хранения и обхода элементов.

8.11.4. Удаление элемента из *Hashtable*

Установка значения в `null` не удаляет элемент из хэша. Для удаления элемента следует использовать метод `Remove`.

```
// Заполняем хэш
Hashtable hash = new Hashtable();
hash.Add(1, "1");
hash.Add(2, "2");
hash.Add(3, "3");

foreach (DictionaryEntry d in hash)
{
  Console.WriteLine("{0}={1}", d.Key, d.Value);
}

// Установка null не удаляет элемент
Console.WriteLine(new string('-', 10));
hash[2] = null;

foreach (DictionaryEntry d in hash)
{
  Console.WriteLine("{0}={1}", d.Key, d.Value);
}

// Метод Remove удаляет элемент
Console.WriteLine(new string('-', 10));
hash.Remove(2);

foreach (DictionaryEntry d in hash)
{
  Console.WriteLine("{0}={1}", d.Key, d.Value);
}
```

8.11.5. Специальный *Hashtable*

Класс `System.Collections.Specialized.CollectionsUtil` позволяет создать специальный объект `Hashtable`, который не различает регистр строк (листинг 8.3).

Листинг 8.3. Объект `Hashtable`, не различающий регистр

```
using System;
using System.Collections;
using System.Collections.Specialized;
```

```
namespace CaseInsensitiveHashtable
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Обычный Hashtable различает регистр
      Hashtable hash1 = new Hashtable();
      hash1["Test"] = 1;
      hash1["test"] = 2;
      Console.WriteLine(hash1["Test"]);
      Console.WriteLine(hash1["test"]);

      // Специальный Hashtable не различает регистр
      Hashtable hash2 =
            CollectionsUtil.CreateCaseInsensitiveHashtable();
      hash2["Test"] = 1;
      hash2["test"] = 2;
      Console.WriteLine(hash2["Test"]);
      Console.WriteLine(hash2["test"]);
    }
  }
}
```

8.12. Сортированный список

Сортированный список позволяет хранить набор пар "ключ/значение" в порядке сортировки по ключу:

```
// Создаем сортированный список
System.Collections.SortedList sl =
              new System.Collections.SortedList();
// Добавляем элементы в произвольном порядке
sl.Add(1, "Один");
sl.Add(5, "Пять");
sl.Add(3, "Три");
sl.Add(2, "Два");
sl.Add(4, "Четыре");
// Перебор всех элементов. Элементы будут выведены
// согласно значению ключа, т. е. в порядке 1..5.
foreach (System.Collections.DictionaryEntry item in sl)
{
  Console.WriteLine(item.Value);
}
```

Для нестандартной сортировки можно реализовать интерфейс `IComparer` (листинг 8.4). Класс `System.Collections.Specialized.CollectionsUtil` позволяет создать специальный объект `SortedList`, который не различает регистр строк.

Листинг 8.4. Реализация `IComparer`

```
using System;
using System.Collections;

namespace Comparer
{
  class Class1
  {
    class Person
    {
      private string name;
      public string Name
      {
        get
        {
          return Name;
        }
      }
      // Конструктор
      public Person(string name)
      {
        this.name = name;
      }
    }
    public class PersonComparer : IComparer
    {
      int IComparer.Compare(Object x, Object y)
      {
        string nx = (x as Person).Name;
        string ny = (y as Person).Name;
        return string.Compare(nx, ny);
      }
    }

    [STAThread]
    static void Main(string[] args)
    {
      ArrayList a = new ArrayList();
```

```
      a.Add(new Person("AAA"));
      a.Add(new Person("CCC"));
      a.Add(new Person("BBB"));

      PersonComparer comp = new PersonComparer();
      a.Sort(comp);

      foreach (Person p in a)
      {
        Console.WriteLine(p.Name);
      }
    }
  }
}
```

8.13. Очередь

Для организации очереди предназначен класс `System.Collections.Queue`:

```
// Создание объекта очереди
System.Collections.Queue queue = new System.Collections.Queue();
// Добавление элементов в очередь
queue.Enqueue(1);
queue.Enqueue(2);
queue.Enqueue(3);

// Выбор элементов из очереди
// Элементы забираются по правилу FIFO (первый зашел — первый вышел)
while (queue.Count > 0)
{
  Console.WriteLine(queue.Dequeue());
}
```

8.14. Стек

Для организации стека используется класс `System.Collections.Stack`:

```
// Создание объекта стека
System.Collections.Stack stack = new System.Collections.Stack();
// Добавление элементов в стек
stack.Push(1);
stack.Push(2);
stack.Push(3);
```

```
// Выбор элементов из стека
// Элементы забираются по правилу FILO
// (первый зашел — последний вышел)
while (stack.Count > 0)
{
  Console.WriteLine(stack.Pop());
}
```

8.15. Преобразование *IList* в *ArrayList*

Метод `ArrayList.Adapter()` позволяет видеть содержимое `IList` (который включает в себя массивы и многие другие коллекции) в виде `ArrayList`, т. е. использовать методы сортировки, поиска и многое другое.

```
int [] array = {1,4,2,3};
System.Collections.ArrayList al =
            System.Collections.ArrayList.Adapter(array);
al.Sort();
foreach (int i in al)
{
  Console.WriteLine(i);
}
```

ГЛАВА 9

Шифрование, кодирование, сжатие, математика

9.1. Вычисление контрольной суммы строки

Вычисление контрольной суммы строки (CRC32) показано в листинге 9.1.

Листинг 9.1. Вычисление контрольной суммы строки

```
using System;
using System.IO;
using System.Text;
namespace CRC32
{
  class Class1
  {
    // Вычисление CRC32
    public static uint Calculate(Stream stream)
    {
      const int BUFFERSIZE = 1024;
      const uint POLYNOMIAL = 0xEDB88320;

      uint result = 0xFFFFFFFF;
      uint Crc32;
      byte[] buffer = new byte[BUFFERSIZE];
      uint[] Crc32Table = new uint[256];
      // Инициализация таблицы
      unchecked
      {
        for (int i = 0; i < 256; i++)
        {
          Crc32 = (uint)i;
```

```
          for (int j = 8; j > 0; j--)
          {
            if ((Crc32 & 1)==1)
              Crc32 = (Crc32 >> 1) ^ POLYNOMIAL;
            else
              Crc32 >>= 1;
          }

          Crc32Table[i] = Crc32;
        }

        // Чтение буфера
        int count = stream.Read(buffer, 0, BUFFERSIZE);
        // Вычисление CRC
        while (count > 0)
        {
          for (int i = 0; i < count; i++)
          {
            result = ((result) >> 8) ^ Crc32Table[(buffer[i]) ^
                     ((result) & 0x000000FF)];
          }

          count = stream.Read(buffer, 0, BUFFERSIZE);
        }
      }

      return ~result;
    }
    [STAThread]
    static void Main(string[] args)
    {
      // Читаем файл test1
      FileStream stream1 = File.OpenRead("test1");
      Console.WriteLine(string.Format("{0:X}", Calculate(stream1)));
      // Читаем файл test2
      FileStream stream2 = File.OpenRead("test2");
      Console.WriteLine(string.Format("{0:X}", Calculate(stream2)));
    }
  }
}
```

9.2. Шифрование строк

Листинг 9.2 содержит пример шифрования строки (или массива байтов) с помощью класса `Rijndael`. Листинги 9.3 и 9.4 демонстрируют пример шифрования строки (или массива байтов) с помощью алгоритма DES.

Листинг 9.2. Шифрование строки с помощью класса `Rijndael`

```
using System;
using System.IO;
using System.Text;
using System.Security.Cryptography;

namespace Crypt_Rijndael
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Исходная строка
      string source_str = "It is string!";
      // Пароль
      string password = "123";

      Console.WriteLine(source_str);

      // Получаем из строки набор байтов, которые будем шифровать
      byte[] source_data = Encoding.UTF8.GetBytes(source_str);
      // Алгоритм
      SymmetricAlgorithm sa_in = Rijndael.Create();
      // Объект для преобразования данных
      ICryptoTransform ct_in = sa_in.CreateEncryptor(
        (new PasswordDeriveBytes(password, null)).GetBytes(16),
         new byte[16]);
      // Поток
      MemoryStream ms_in = new MemoryStream();
      // Шифровальщик потока
      CryptoStream cs_in =
           new CryptoStream(ms_in, ct_in, CryptoStreamMode.Write);
      // Записываем шифрованные данные в поток
      cs_in.Write(source_data, 0, source_data.Length);
      cs_in.FlushFinalBlock();
      // Создаем строку
      string crypt_str = Convert.ToBase64String(ms_in.ToArray());
      // Выводим зашифрованную строку
      Console.WriteLine(crypt_str);

      // Получаем массив байтов
      byte[] crypt_data = Convert.FromBase64String(crypt_str);
```

```
      // Алгоритм
      SymmetricAlgorithm sa_out = Rijndael.Create();
      // Объект для преобразования данных
      ICryptoTransform ct_out = sa_out.CreateDecryptor(
        (new PasswordDeriveBytes(password, null)).GetBytes(16),
        new byte[16]);
      // Поток
      MemoryStream ms_out = new MemoryStream(crypt_data);
      // Расшифровываем поток
      CryptoStream cs_out =
           new CryptoStream(ms_out, ct_out, CryptoStreamMode.Read);
      // Создаем строку
      StreamReader sr_out = new StreamReader(cs_out);
      string source_out = sr_out.ReadToEnd();
      // Выводим расшифрованную строку
      Console.WriteLine(source_out);
    }
  }
}
```

Листинг 9.3. Шифрование строки с помощью алгоритма DES

```
using System;
using System.IO;
using System.Text;
using System.Security.Cryptography;

namespace Crypt_DES
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Исходная строка
      string source_str = "It is string!";

      Console.WriteLine(source_str);

      // Получаем из строки набор байтов, которые будем шифровать
      byte[] source_data = Encoding.UTF8.GetBytes(source_str);

      DES des = new DESCryptoServiceProvider();
```

```
      // Генерируем IV и Key
      des.GenerateIV();
      des.GenerateKey();

      // Сохраняем ключи
      byte[] IVByteArray  = des.IV;
      byte[] KeyByteArray = des.Key;

      // Поток выходных данных
      MemoryStream ms_in = new MemoryStream();
      // Шифрованный поток
      CryptoStream cs_in =
        new CryptoStream(ms_in,
        des.CreateEncryptor(KeyByteArray,
        IVByteArray),
        CryptoStreamMode.Write);
      cs_in.Write(source_data, 0, source_data.Length);
      cs_in.Close();
      // Получаем зашифрованную строку
      string crypt_str = Convert.ToBase64String(ms_in.ToArray());
      // Выводим зашифрованную строку
      Console.WriteLine(crypt_str);

      // Получаем массив байтов
      byte[] crypt_data = Convert.FromBase64String(crypt_str);
      // Поток выходных данных
      MemoryStream ms_out = new MemoryStream(crypt_data);
      // Поток для расшифровки
      CryptoStream cs_out =
        new CryptoStream(ms_out,
        des.CreateDecryptor(KeyByteArray,
        IVByteArray),
        CryptoStreamMode.Read);
      // Читаем выходную строку
      StreamReader sr_out = new StreamReader(cs_out);
      string source_out = sr_out.ReadToEnd();
      // Выводим расшифрованную строку
      Console.WriteLine(source_out);
    }
  }
}
```

Листинг 9.4. Шифрование строки с помощью алгоритма DES3

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace Crypt_DES1
{
  class DES_Test
  {
    static void Main()
    {
      string       original, encrypted, decrypted, password;
      TripleDESCryptoServiceProvider des;
      MD5CryptoServiceProvider  provider;
      byte[]       pwdhash, buff;

      // Пароль
      password = "secretpassword1!";

      // Исходная строка
      original = "test string";

      // Генерация MD5-хэша пароля
      provider = new MD5CryptoServiceProvider();
      pwdhash = provider.ComputeHash(
               ASCIIEncoding.ASCII.GetBytes(password));
      provider = null;

      //-----------------------------------------------
      // Шифрование
      //-----------------------------------------------
      // DES3-шифрование
      des = new TripleDESCryptoServiceProvider();
      // Ключ для DES - это хэш пароля
      des.Key = pwdhash;
      // Режим шифрования
      des.Mode = CipherMode.ECB; // CBC, CFB

      // Преобразуем строку в массив байтов
      buff = ASCIIEncoding.ASCII.GetBytes(original);

      // Шифруем
      encrypted = Convert.ToBase64String(
            des.CreateEncryptor().TransformFinalBlock(buff, 0,
                                    buff.Length)
        );
```

```
      //-----------------------------------------------
      // Расшифровка
      //-----------------------------------------------
      buff = Convert.FromBase64String(encrypted);
      decrypted = ASCIIEncoding.ASCII.GetString(
        des.CreateDecryptor().TransformFinalBlock(buff, 0,
                                          buff.Length)
        );

      // Освобождаем ресурс
      des = null;

      // Печатаем результат
      Console.WriteLine("Исходная строка  : {0}", original);
      Console.WriteLine("Зашифрованная строка : {0}", encrypted);
      Console.WriteLine("Расшифрованная строка: {0}", decrypted);

    }
  }
}
```

9.3. Контроль арифметических операций

Операторы `checked` и `unchecked` позволяют управлять контролем арифметических операций. Например, вычисление байтовой контрольной суммы с переполнением может выглядеть следующим образом:

```
byte CS = 0;
// Вычислить контрольную сумму с переполнением.
// Для отключения контроля переполнения используем
// оператор unchecked
unchecked
{
  foreach (byte value in array)
  {
    CS += value;
  }
}
```

Кроме того, оба оператора можно использовать как унарные арифметические функции, что в некоторых случаях более предпочтительно:

```
A = unchecked(B * C);
```

9.4. Вычисление математических выражений

Для работы приведенного далее кода требуется подключение библиотек `Microsoft.JScript` и `Microsoft.Vsa`:

```
object result = Microsoft.JScript.Eval.JScriptEvaluate("2+2",
                 Microsoft.JScript.Vsa.VsaEngine.CreateEngine());
Console.WriteLine(result);
```

ГЛАВА 10

Строковые и символьные операции

10.1. Проверка символов

Класс `char` имеет несколько полезных статических методов для проверки символов:

- `char.IsLetter(char c)` — возвращает `true`, если символ является буквой;
- `char.IsDigit(char c)` — возвращает `true`, если символ является цифрой;
- `char.IsControl(char c)` — возвращает `true`, если символ является управляющим символом (например, знаком табуляции `\t`);
- `char.IsLetterOrDigit(char c)` — возвращает `true`, если символ является буквой или цифрой;
- `char.IsLower(char c)` — возвращает `true`, если символ является символом нижнего регистра;
- `char.IsPunctuation(char c)` — возвращает `true`, если символ является знаком пунктуации (например, точкой или запятой);
- `char.IsSeparator(char c)` — возвращает `true`, если символ является разделителем;
- `char.IsUpper(char c)` — возвращает `true`, если символ является символом верхнего регистра.

10.2. Создание строки одинаковых символов

Один из конструкторов класса `string` позволяет создать строку из одинаковых символов:

```
Console.WriteLine(new string('-', 20));
```

10.3. Проверка, является ли строка числом

Для проверки строки можно использовать метод `Parse`, заключенный в блок `try-catch`:

```
int value;
try
{
   value = int.Parse(someString);
}
catch (Exception e)
{
  // Нет, строка не является числом типа int
}
```

Некоторые классы (например, `float`) поддерживают метод `TryParse`, позволяющий обойтись без медлительной операции генерации исключения.

10.4. Проверка, является ли строка допустимой датой

Для проверки допустимости даты можно использовать метод `System.DataTime.Parse`:

```
string strMyDateTime = "11/01/04";
System.DateTime myDateTime;
bool isValid = true;
try
{
  myDateTime = System.DateTime.Parse(strMyDateTime);
}
catch (Exception e)
{
  isValid = false;
}
```

10.5. Разбиение строки по разделителям

Метод `Split` позволяет разбить строку на части и получить массив строк:

```
string InputStr;
// Открыть файл для чтения
using (StreamReader reader = new StreamReader("input.txt"))
```

```
{
  // Прочитать весь файл
  InputStr = reader.ReadToEnd();
}

// Список разделителей файла
static char [] Delimeters =
new char [] {'.',',',';','!',' ','\n','\r','-','=','\\','/'};

// Разделить файл на слова
string [] Words = InputStr.Split(Delimeters);
```

10.6. Объединение строк через разделитель

Для объединения строк через разделитель вместо цикла по всем элементам массива можно использовать метод `Join` класса `string`:

```
// Массив строк
string[] val = {"apple", "orange", "grape", "pear"};
// Объединяем строки через запятую
string result= string.Join(",", val);
// Результат: <apple,orange,grape,pear>
Console.WriteLine(result);
```

С помощью этого же метода можно объединять некоторый диапазон строк массива:

```
// Объединение трех элементов, начиная со второго
string result= string.Join(",", val, 2, 3);
```

При необходимости объединить строки без разделителя можно использовать метод `string.Concat`:

```
string result= string.Concat(val);
```

10.7. Какие преимущества дает класс *StringBuilder*?

Основное преимущество класса `StringBuilder` в том, что этот класс умеет повторно использовать уже отведенную память. При обычном сложении строк память будет отводиться при каждой операции, тогда как при использовании `StringBuilder` этих затрат не будет. Пример использования `StringBuilder` для объединения строк:

```
StreamReader sr = File.OpenText(dlg.FileName);
string s = sr.ReadLine();
```

```
StringBuilder sb = new StringBuilder();
while (s != null)
{
  sb.Append(s);
  s = sr.ReadLine();
}
sr.Close();
textBox1.Text = sb.ToString();
```

Выигрыш по времени получается весьма существенный. Так, например, конкатенация 50 000 строк занимает примерно минуту, а при использовании `StringBuilder` та же операция выполняется около секунды (о замерах времени выполнения *см. разд. 5.7*). Кроме того, не стоит забывать, что вместо `sb.Append(string.Format())` можно использовать `sb.AppendFormat`.

10.8. Как сравнить все строки без учета регистра символов?

Сравнение можно выполнить с помощью метода `System.String.Compare`:

```
"book" == "Book"                              // условие равно false
System.String.Compare("book", "Book", true) // условие равно true
```

10.9. Прямая модификация содержимого строки

Преобразование строки к верхнему регистру:

```
public static unsafe void ToUpper(string str)
{
  fixed (char *pfixed = str)
  for (char *p=pfixed; *p != 0; p++)
    *p = char.ToUpper(*p);
}
```

Преимущество этого метода — при преобразовании не создается копия строки, а используется память, занятая самой строкой. Недостаток — необходимость использования незащищенного кода.

10.10. Специальные символы в строке

В C# определены следующие специальные символы (escape-последовательности):

- `\'` — одинарная кавычка;
- `\"` — двойная кавычка;

- `\\` — обратный слэш;
- `\0` — нулевой символ Unicode;
- `\a` — звуковой сигнал (Alert);
- `\b` — символ Backspace;
- `\f` — перевод страницы (Form feed);
- `\n` — перевод строки;
- `\r` — возврат каретки;
- `\t` — горизонтальная табуляция;
- `\uxxxx` — символ Unicode с указанным шестнадцатеричным кодом;
- `\x`*n*`[`*n*`][`*n*`][`*n*`]` — символ Unicode с указанным кодом;
- `\Uxxxxxxxx` — символ Unicode с указанным кодом.

Отличие трех последних символов заключается в том, что символы `\u` и `\U` требуют обязательного указания 4 и 8 знаков кода соответственно, а для `\x` можно указать произвольное количество знаков кода:

- разрешенная запись: `\u0024`, `\x024`, `\x24`, `\U00000024`;
- некорректная запись (ошибка компиляции "неразрешенная последовательность символов"): `\u24`, `\u024`, `\U0024`.

ГЛАВА 11

Консольные программы

11.1. Вывод цветного текста

Для вывода цветного текста на консоль можно использовать класс, приведенный в листинге 11.1.

В C# 2.0 для вывода цветного текста можно использовать свойства `Console.BackgroundColor` и `Console.ForegroundColor`. Для восстановления цвета по умолчанию предназначен метод `Console.ResetColor()`.

```
Console.BackgroundColor = ConsoleColor.Black;
Console.ForegroundColor = ConsoleColor.Blue;
Console.Write("Цветной текст");
Console.ResetColor();
```

Листинг 11.1. Вывод цветного текста на консоль

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

namespace ColorText
{
  public class ConsoleColorManager
  {
    private Handles m_Handle;

    // Константы консольных потоков
    public enum Handles
    {
      STD_INPUT_HANDLE  = -10,
```

```
  STD_OUTPUT_HANDLE = -11,
  STD_ERROR_HANDLE  = -12
}

// Импортируем метод GetStdHandle
[DllImportAttribute("Kernel32.dll")]
private static extern IntPtr GetStdHandle
(
  int nStdHandle      // тип консольного потока
);

// Импортируем метод SetConsoleTextAttribute
[DllImportAttribute("Kernel32.dll")]
private static extern bool SetConsoleTextAttribute
(
  IntPtr hConsoleOutput,  // дескриптор потока
  int wAttributes         // цвет текста и фона
);

// Цвета (флаги)
[Flags]
public enum Color
{
  Black   = 0x0000,
  Blue    = 0x0001,
  Green   = 0x0002,
  Cyan    = 0x0003,
  Red     = 0x0004,
  Magenta = 0x0005,
  Yellow  = 0x0006,
  Grey    = 0x0007,
  White   = 0x0008
}

// По умолчанию будем управлять потоком OUTPUT
public ConsoleColorManager()
{
  m_Handle = Handles.STD_OUTPUT_HANDLE;
}

public ConsoleColorManager(Handles Handle)
{
  m_Handle = Handle;
}
```

```
    // Установка цвета по умолчанию
    public bool SetDefaultColor()
    {
      return SetColor(Color.Grey, true);
    }

    // Установка цвета текста.
    // Флаг fLight задает яркость цвета
    public bool SetColor(Color aColor, bool fLight)
    {
      // получаем дескриптор
      IntPtr ConsoleHandle = GetStdHandle((int)m_Handle);

      // получаем значение цвета
      int colorMask = (int)aColor;

      // Если нужен яркий цвет, добавляем White
      if (fLight)
        colorMask |= (int)Color.White;

      // Устанавливаем
      return SetConsoleTextAttribute(ConsoleHandle, colorMask);
    }
  }

  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      ConsoleColorManager colorManager = new ConsoleColorManager();
      // Ярко-зеленый
      colorManager.SetColor(ConsoleColorManager.Color.Green, true);
      Console.WriteLine("Text in green");

      // Темно-красный
      colorManager.SetColor(ConsoleColorManager.Color.Red  , false);
      Console.WriteLine("Text in red");

      // Восстанавливаем цвет по умолчанию
      colorManager.SetDefaultColor();
    }
  }
}
```

11.2. Задержка закрытия консольной программы

Добавление вызова `Console.ReadLine()` позволяет задержать завершение консольной программы до нажатия клавиши <Enter>. Иначе окно консольной программы закрывается сразу же после завершения программы.

При добавлении условной компиляции нажатие клавиши <Enter> будет требоваться только в режиме отладки:

```
#if DEBUG
   Console.ReadLine();
#endif
```

В Visual Studio 2005 эта проблема решена — консольная программа, запущенная из среды разработки, не закроется без нажатия любой клавиши.

11.3. Перенаправление вывода консольной программы

Вывод консоли является потоком, который можно перенаправить, например, в файл (листинг 11.2).

Листинг 11.2. Перенаправление вывода консольной программы

```
using System;
using System.IO;
using System.Text;
namespace ConsoleOutput
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Файл для вывода
      FileStream file = new FileStream("Test.txt", FileMode.Create);
      // Сохраняем стандартную консоль
      TextWriter out_save = Console.Out;
      // Устанавливаем файл для вывода данных
      TextWriter out_file = new StreamWriter(file);
      Console.SetOut(out_file);
      // Пробуем выводить в файл
      Console.WriteLine("write to file");
```

```
      // Восстанавливаем стандартную консоль
      Console.SetOut(out_save);
      Console.WriteLine("write to console");

      // Закрываем файл
      out_file.Close();
    }
  }
}
```

11.4. Изменение заголовка консольной программы

С помощью метода `SetConsoleTitle` из библиотеки `kernel32` можно изменить заголовок консольной программы (листинг 11.3). Посредством метода `GetConsoleTitle` можно прочитать заголовок окна консольной программы.

В C# 2.0 заголовок окна можно изменить с помощью свойства `Console.Title`.

Листинг 11.3. Установка заголовка консольной программы

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

namespace ConsoleTitle
{
  class Class1
  {
    [DllImport("kernel32.dll")]
    public static extern bool
                SetConsoleTitle(String lpConsoleTitle); );

    [DllImport("KERNEL32.DLL", EntryPoint="GetConsoleTitleA",
                                CharSet=CharSet.Ansi)]
    internal static extern int GetConsoleTitle(
       StringBuilder lpConsoleTitle, int nSize);

    [STAThread]
    static void Main(string[] args)
    {
      SetConsoleTitle("New title");
      Console.ReadLine();
    }
  }
}
```

11.5. Получение размера окна консольной программы

Получить размер окна консольной программы можно посредством метода `GetConsoleScreenBufferInfo` (листинг 11.4). В C# 2.0 размер окна можно получить и изменить с помощью полей `Console.WindowHeight` и `Console.WindowWidth`.

Листинг 11.4. Получение размера окна консольной программы

```
[StructLayout(LayoutKind.Sequential)]
internal struct CONSOLE_SCREEN_BUFFER_INFO
{
  public COORD dwSize;
  public COORD dwCursorPosition;
  public short wAttributes;
  public SMALL_RECT srWindow;
  public COORD dwMaximumWindowSize;
}

[DllImport("KERNEL32.DLL", EntryPoint="GetConsoleScreenBufferInfo")]
internal static extern int GetConsoleScreenBufferInfo
 (
     int hConsoleOutput,
     ref CONSOLE_SCREEN_BUFFER_INFO lpConsoleScreenBufferInfo);

CONSOLE_SCREEN_BUFFER_INFO SBI = new CONSOLE_SCREEN_BUFFER_INFO();
GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HANDLE),
                           ref SBI);

WindowWidth = SBI.srWindow.Right - SBI.srWindow.Left + 1;
WindowHeight = SBI.srWindow.Bottom - SBI.srWindow.Top + 1;
```

ГЛАВА 12

Производительность

12.1. Не кэшируйте соединение с БД

Оптимальный путь соединения с БД — создание и уничтожение объектов соединения каждый раз, когда это необходимо. Кэширование соединения — плохая стратегия в любых вариантах:

- если несколько страниц используют одно соединение, закэшированное в объекте `Application`, то каждая страница будет бороться за использование этого соединения;
- если соединение закэшировано в объекте `Session`, то соединение с БД будет создано для каждого клиента, что приводит к загрузке сервера и БД.

Создание и уничтожение объектов на каждой странице приложения является эффективным вариантом работы, т. к. IIS имеет специальный пул соединений, эффективно управляющий созданием и освобождением соединений с БД.

Именно так делается в примерах из Microsoft Data Access Application Block:

```
using (SqlConnection connection =
                        new SqlConnection(connectionString))
{
  connection.Open();
  // далее - операторы, использующие созданное соединение
}
```

12.2. Используйте *StringBuilder*

Строки в C# являются константными объектами. Это означает, что операция

```
string s = s1 + s2;
```

вызывает создание новой строки и копирование в нее содержимого строк `s1` и `s2`. Это очень длительная операция. Избежать ее помогает класс `StringBuilder`:

```
StringBuilder sb = new StringBuilder();
for (int i=0; i<1000; i++)
{
  sb.Append(s);
}
textBox1.Text = sb.ToString();
```

Выигрыш по времени весьма существенный. Так, например, конкатенация 50 000 строк занимает примерно минуту, а при использовании `StringBuilder` та же операция выполняется около секунды (о замерах времени выполнения *см. разд. 5.7*).

Кроме того, не стоит забывать, что вместо `sb.Append(string.Format())` можно использовать `sb.AppendFormat`.

12.3. Используйте *Length* для проверки пустоты строки

Можно придумать множество способов проверить пустоту строки:

```
if (myString == "")
if (myString == String.Empty)
if (myString.Length == 0)
if (String.Equals(myString, String.Empty))
```

Оптимальный способ проверки на пустую строку — сравнение значения `Length` с нулем (третий вариант), но следует помнить, что этот способ работает, только если `myString` не может принимать значение `null`. Если `myString` может принимать значение `null`, то необходимо использовать метод `Equals` (четвертый вариант).

Ян Нельсон (Ian Nelson) приводит такие оценки производительности (табл. 12.1).

Таблица 12.1. *Оценки производительности*

Конструкция	true	false
`myString==String.Empty`	1172	2484
`myString.Length==0`	531	531
`String.Equals(myString,String.Empty)`	611	1893

12.4. Не используйте исключения, если это возможно

Генерация исключения является чрезвычайно длительной операцией. Не следует применять исключения, если без них можно обойтись. Например, код

```
try
{
  return b/c;
}
catch (DivideByZeroException)
{
  return 0;
}
```

вполне можно переписать без использования исключений:

```
if (c == 0)
  return 0;
return b/c;
```

12.5. Используйте *AddRange* для добавления групп элементов

Вместо многократного добавления каждого элемента коллекции по отдельности, оптимальнее использовать `AddRange`, чтобы добавить сразу всю коллекцию (автор Виктор Шатохин (Viktor Shatokhin)). Практически все элементы управления Windows и коллекции (например, `StringCollection`, `TraceCollection`, `HttpWebRequest`, `UserControlm ColumnHeader` и т. д.) имеют и метод `Add`, и метод `AddRange`, и каждый из них оптимизирован для разных целей. `Add` служит для добавления отдельного элемента, в то время как `AddRange` приводит к некоторым потерям производительности, но приносит выигрыш при добавлении множества элементов.

12.6. Создавайте *Hashtable* и *ArrayList* подходящего размера

В конструкторе `Hashtable` и `ArrayList` можно указывать начальный размер набора данных. При выходе за начальные границы эти объекты автоматически увеличивают свой размер, но следует помнить, что эта операция довольно длительная. Гораздо выгоднее заранее указать необходимый размер.

12.7. Иногда выгоднее использовать невыровненный массив

Работа с невыровненными массивами происходит быстрее, чем с обычными. Например, код

```
for (int i=0; i<size; i++)
  for (int j=0; j<size; j++)
    a1[i][j] = i+j;
```

будет выполняться почти в два раза быстрее, чем

```
for (int i=0; i<size; i++)
  for (int j=0; j<size; j++)
    a2[i,j] = i+j;
```

Однако следует помнить о том, что эти массивы нужно создать, тогда как код

```
int [][] a1 = new int[size][];
  for (int i=0; i<size; i++)
    a1[i] = new int[size];
```

выполняется значительно медленнее, чем

```
int [,] a2 = new int[size, size];
```

Если учитывать время создания массивов, то обычный массив значительно эффективнее, а если только время заполнения — то выигрывает невыровненный массив.

12.8. Используйте *DataReader* для последовательного доступа к данным

Класс `DataReader` дает значительный выигрыш в производительности. Если нет необходимости кэшировать данные и достаточно простого последовательного чтения, то лучше использовать `DataReader`. При необходимости кэширования или сложной обработки данных применяется класс `DataSet`. Разумеется, `DataReader` не может быть передан через Web-сервис. Кроме того, `DataReader` не может быть привязан к нескольким объектам одновременно, т. к. он умеет читать данные только вперед.

12.9. Используйте хранимые процедуры

Использование хранимых процедур позволяет сэкономить на трафике и времени выполнения запросов, т. к. на сервер передаются только данные, а сами процедуры уже скомпилированы.

ГЛАВА 13

Списки файлов, каталогов

13.1. Получение списка логических дисков

Для получения списка логических дисков можно использовать метод `GetLogicalDrives` (листинг 13.1).

Еще один способ — вызов метода класса `Environment`:

```
string[] LogicalDrives = Environment.GetLogicalDrives();
foreach(string driver in LogicalDrives)
{
  Console.WriteLine(driver);
}
```

Следует обратить внимание, что при отсутствии прав на выполнение этой операции первый метод генерирует исключение `UnauthorizedAccessException`, а второй — `SecurityException`.

Листинг 13.1. Получение списка логических дисков

```
using System;
namespace GetLogicalDrives
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      GetLogicalDrives();
      Console.ReadLine();
    }
```

```
    static void GetLogicalDrives()
    {
      try
      {
        string[] drives = System.IO.Directory.GetLogicalDrives();
        foreach (string driver in drives)
        {
          System.Console.WriteLine(driver);
        }
      }
      catch (System.IO.IOException)
      {
        System.Console.WriteLine("Ошибка (I/O error)");
      }
      catch (System.Security.SecurityException)
      {
        System.Console.WriteLine(
               "Недостаточно прав для выполнения операции");
      }
    }
  }
}
```

13.2. Получение списка сетевых дисков

Пример получения списка сетевых дисков приводится в разд. 15.8.

13.3. Список каталогов

Получить список каталогов можно с помощью метода `GetDirectories` класса `System.IO.Directory` (листинг 13.2).

Листинг 13.2. Получение списка каталогов

```
try
{
  string[] dirs = Directory.GetDirectories(@"c:\");
  Console.WriteLine("Всего каталогов: {0}.", dirs.Length);
  foreach (string dir in dirs)
  {
    Console.WriteLine(dir);
  }
}
```

```
catch (Exception e)
{
  Console.WriteLine("Ошибка: {0}", e.ToString());
}
```

13.4. Список каталогов по маске

Получить список каталогов по маске можно с помощью одной из реализаций метода `GetDirectories` класса `System.IO.Directory` (листинг 13.3).

Листинг 13.3. Получение списка каталогов, начинающихся на букву p

```
try
{
  // Получить список каталогов на букву p
  string[] dirs = Directory.GetDirectories(@"c:\", "p*");
  Console.WriteLine("Всего каталогов на букву p: {0}.",
                    dirs.Length);
  foreach (string dir in dirs)
  {
    Console.WriteLine(dir);
  }
}
catch (Exception e)
{
  Console.WriteLine("Ошибка: {0}", e.ToString());
}
```

13.5. Список файлов

Получить список файлов в заданном каталоге можно с помощью метода `GetFiles` класса `System.IO.Directory` (листинг 13.4).

Листинг 13.4. Получение списка файлов

```
try
{
  // Список файлов диска C:
  string[] dirs = Directory.GetFiles(@"c:\");
  Console.WriteLine("Всего файлов {0}.", dirs.Length);
  foreach (string dir in dirs)
  {
  Console.WriteLine(dir);
  }
}
```

```
catch (Exception e)
{
  Console.WriteLine("Ошибка: {0}", e.ToString());
}
```

13.6. Список файлов по маске

Получить список файлов по маске можно с помощью метода `GetFiles` класса `System.IO.Directory` (листинг 13.5).

Листинг 13.5. Получение списка файлов

```
try
{
  // Список файлов с расширением log диска C:
  string[] dirs = Directory.GetFiles(@"c:\", "*.log");
  Console.WriteLine("Всего файлов {0}.", dirs.Length);
  foreach (string dir in dirs)
  {
    Console.WriteLine(dir);
  }
}
catch (Exception e)
{
  Console.WriteLine("Ошибка: {0}", e.ToString());
}
```

Еще один способ — использование класса `System.IO.DirectoryInfo`:

```
// Создаем DirectoryInfo для текущего каталога
DirectoryInfo dir = new DirectoryInfo(".");
// Перебираем все файлы с расширением cs
foreach (FileInfo f in dir.GetFiles("*.cs"))
{
  string name = f.FullName;                // имя файла
  long size = f.Length;                    // размер
  DateTime creationTime = f.CreationTime;  // время создания
  Console.WriteLine("{0,-12:N0} {1,-20:g} {2}", size,
                creationTime, name);
}
```

13.7. Создание временного файла

Уникальное имя для временного файла в каталоге, определенном переменной окружения `TEMP`, можно получить через метод `System.IO.Path.GetTempFileName()`.

ГЛАВА 14

Файловые операции

14.1. Блокирование ошибки "Устройство не готово"

При попытке вызвать перечисление файлов с отсутствующего или не готового к работе устройства (например, с дисковода, при отсутствии дискеты), Windows выдает окно с сообщением "Устройство не готово". Для того чтобы изменить поведение Windows, необходимо воспользоваться Win32-функцией `SetErrorMode`, как показано в листинге 14.1.

Листинг 14.1. Использование `SetErrorMode`

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

namespace ChangeErrorModeTest
{
  class GetFilesClass
  {
    [Flags]
      public enum ErrorModes
    {
      Default = 0x0,
      FailCriticalErrors = 0x1,
      NoGpFaultErrorBox = 0x2,
      NoAlignmentFaultExcept = 0x4,
      NoOpenFileErrorBox = 0x8000
    }
```

```
    public struct ChangeErrorMode : IDisposable
    {
      private int _oldMode;

      public ChangeErrorMode(ErrorModes mode)
      {
        _oldMode = SetErrorMode((int)mode);
      }

      void IDisposable.Dispose() { SetErrorMode(_oldMode); }

      [DllImport("kernel32.dll")]
      private static extern int SetErrorMode(int newMode);
    }

    [STAThread]
    static void Main(string[] args)
    {
      using(new ChangeErrorMode(ErrorModes.FailCriticalErrors))
      {
        try
        {
          string [] files = System.IO.Directory.GetFiles(@"A:\");
          foreach (string file in files)
          {
            Console.WriteLine(file);
          }
        }
        catch
        {
          Console.WriteLine("Устройство не готово");
        }
      }
    }
  }
}
```

14.2. Получение списка файлов и каталогов

Для получения списка и файлов и каталогов можно использовать функцию `GetFileSystemEntries`:

```
string[] contents =
            System.IO.Directory.GetFileSystemEntries(@"E:\");
foreach(string s in contents)
  Console.WriteLine(s);
```

14.3. Диалог открытия папки

Стандартный диалог Windows для открытия папки можно вызывать с использованием функции библиотеки `Shell32`. Для этого необходимо подключить COM-объект **Microsoft Shell Controls And Automation** (модуль `shell32.Dll`). Вызов диалога выбора папки выглядит следующим образом:

```
using Shell32;
  Shell shell = new Shell();
  Folder folder = shell.BrowseForFolder(
    0,                    // Hwnd
    "Выберите папку...", // заголовок окна
    0,                    // Параметры
    ShellSpecialFolderConstants.ssfDESKTOP // начальная папка
  );
```

Класс `Folder` имеет довольно обширный список методов. Например, получение списка всех файлов и каталогов выполняется так:

```
if (folder != null)
{
  foreach (FolderItem fi in folder.Items())
  {
    Console.WriteLine(fi.Name);
  }
}
```

14.4. Основные операции с каталогами

Операции с каталогами выполняются с помощью методов класса `System.IO.Directory`:

- `DirectoryInfo CreateDirectory(string` *`path`*`)` — создание каталога;
- `void Move(string` *`FromDir`*`, string` *`ToDir`*`)` — переименование или перемещение каталога;
- `void Delete(string` *`path`*`)` — удаление каталога;
- `void Delete(string` *`path`*`, bool` *`recursive`*`)` — если параметр `recursive` равен `true`, то удаление каталога со всеми файлами и подкаталогами;
- `bool Exists(string` *`path`*`)` — возвращает `true`, если путь существует.

14.5. Основные операции с файлами

Операции с файлами выполняются с помощью методов класса `System.IO.FileInfo`:

- `bool Exists` — возвращает `true`, если файл существует;
- `FileInfo CopyTo(string ToFileName)` — копирует файл;
- `FileInfo CopyTo(string ToFileName, bool overwrite)` — если *overwrite* равно `true`, то копирует файл, переписывая существующий;
- `FileStream Create()` — создает файл;
- `StreamWriter CreateText()` — создает текстовый файл;
- `void Delete()` — удаляет файл;
- `void MoveTo(string NewFileName)` — перемещает файл.

Для выполнения операций с файлом необходимо создать экземпляр класса `FileInfo` и вызвать соответствующий метод:

```
System.IO.FileInfo file = new System.IO.FileInfo(@"E:\tmp1");
file.Delete();
```

14.6. Чтение и установка атрибутов файла

Для работы с атрибутами файлов используется класс `FileAttributes` и методы `GetAttributes()`, `SetAttributes()` класса `File`:

```
FileAttributes attrib = File.GetAttributes(FileName);
if ((attrib & FileAttributes.ReadOnly) == FileAttributes.ReadOnly)
{
  throw new Exception("файл имеет атрибут только для чтения!");
}
```

Получить атрибуты файла можно и с помощью класса `System.IO.FileInfo`:

```
System.IO.FileInfo file = new System.IO.FileInfo(@"E:\prop.txt");
Console.WriteLine(file.Attributes);
```

14.7. Создание и чтение бинарного файла

Для создания бинарного файла можно использовать класс `BinaryWriter`:

```
// Создаем бинарный поток для записи файла
FileStream fs = new FileStream(FileName, FileMode.CreateNew);
BinaryWriter w = new BinaryWriter(fs);
```

```
// Записываем данные в файл
for (int i = 0; i < 11; i++)
{
  w.Write( (int) i);
}
// Закрываем файл
w.Close();
// fs.Close();  не требуется, будет вызван автоматически!
```

Чтение бинарного файла производится с помощью класса `BinaryReader`:

```
// Создание бинарного потока для чтения файла
fs = new FileStream(FileName, FileMode.Open, FileAccess.Read);
BinaryReader r = new BinaryReader(fs);
// Читаем данные из файла
for (int i = 0; i < 11; i++)
{
  Console.WriteLine(r.ReadInt32());
}
// Закрываем поток
r.Close();
// fs.Close();  не требуется, будет вызван автоматически!
```

14.8. Создание текстового файла

Создание текстового файла проще всего выполнить с помощью класса `StreamWriter`:

```
using (StreamWriter sw = new StreamWriter("TestFile.txt"))
{
  // Добавляем строки файла
  sw.Write("Первая");
  sw.WriteLine("строка файла.");
  sw.WriteLine("Вторая строка файла");
}
```

Кроме того, можно использовать метод `CreateText` класса `System.IO.File`:

```
StreamWriter sr = File.CreateText(FILE_NAME);
sr.WriteLine ("Первая строка");
sr.WriteLine ("Можно записать в файл int {0} или floats {1}",
              1, 4.2);
sr.Close();
```

Отличие первого способа от второго заключается еще и в использовании оператора `using`. В первом случае закрытие потока произойдет автоматически при выходе за границу жизни `sw`, т. е. при выходе из границы `using`. Во вто-

ром случае требуется обязательный вызов метода `Close()`. При записи больших объемов данных может потребоваться вызов метода `Flush()`.

14.9. Добавление в текстовый файл

Для добавления текста в текстовый файл предназначен метод `AppendText` класса `StreamWriter`, экземпляр которого возвращает метод `File.AppendText`:

```
StreamWriter sw;
sw=File.AppendText("C:\\MyTextFile.txt");
sw.WriteLine("new line");
sw.Close();
```

14.10. Чтение и запись в файл строк на русском языке

При записи строк на русском языке необходимо указывать соответствующую кодировку:

```
System.IO.StreamWriter sw=
   new System.IO.StreamWriter("test.txt",false,
         System.Text.Encoding.GetEncoding(1251));
sw.WriteLine("Строка");
sw.Close();
```

Для чтения аналогично используется класс `System.IO.StreamReader`.

14.11. Посимвольное чтение текстового файла

Посимвольное чтение текстового файла (листинг 14.2) осуществляется с помощью метода `Read()` класса `StreamReader`. Если достигнут конец файла, метод возвращает –1. Метод `Peek()` выполняет те же действия, что и `Read()`, но при этом не передвигает указатель файла. Класс `StreamReader` находится в пространстве имен `System.IO`.

Листинг 14.2. Использование класса `StreamReader` для посимвольного чтения файла

```
using System;
using System.IO;

namespace ReadFile1
{
  class Class1
```

```
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Создать объект для чтения файла
      StreamReader reader;
      try
      {
        reader = new StreamReader("input.txt");
      }
      catch{
        // При открытии файла получили исключение
        Console.WriteLine("Ошибка открытия файла");
        return;
      }
      // Посимвольное чтение файла
      int ch;
      while ((ch = reader.Read()) != -1)
      {
        Console.WriteLine(ch);
      }
    }
  }
}
```

14.12. Построчное чтение текстового файла

Построчное чтение текстового файла осуществляется с помощью метода `ReadLine()` класса `System.IO.StreamReader`. Если достигнут конец файла, возвращает `null`.

```
FileStream fs =
  new FileStream( @"c:\test.txt", FileMode.Open, FileAccess.Read);
StreamReader sr = new StreamReader( fs );
string curLine;
while( (curLine = sr.ReadLine()) != null )
  Console.WriteLine( curLine );
sr.Close();
```

14.13. Чтение файла полностью

Текстовый файл можно полностью загрузить в строку с помощью метода `ReadToEnd()` класса `System.IO.StreamReader`. Если достигнут конец файла, возвращает `null`.

14.14. Отслеживание изменений в файловой системе

Класс `System.IO.FileSystemWatcher` позволяет следить за изменениями в файловой системе. В частности, можно отслеживать изменения в файлах и подкаталогах указанного каталога. При этом можно отслеживать файлы не только на локальном компьютере, но и на сетевых дисках или удаленных компьютерах. Листинг 14.3 демонстрирует пример использования этого класса.

Листинг 14.3. Отслеживание изменений в файловой системе

```
using System;
using System.IO;

namespace FileSystemWatcherTest
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Создаем наблюдателя
      FileSystemWatcher watcher = new FileSystemWatcher();
      watcher.Path = @"E:\1\";
      // Будем следить за изменениями по последнему доступу,
      // времени записи и переименованию файла или каталога
      watcher.NotifyFilter =
          NotifyFilters.LastAccess | NotifyFilters.LastWrite |
          NotifyFilters.FileName | NotifyFilters.DirectoryName;
      // Будем следить только за txt-файлами
      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);

      // Включаем наблюдение
      watcher.EnableRaisingEvents = true;

      // Ждем, пока пользователь не нажмет клавишу <q>
      Console.WriteLine("Нажмите \'q\' для выхода.");
```

```
      while(Console.Read()!='q');
    }

    private static void OnChanged(object source,
                                  FileSystemEventArgs e)
    {
      // Файл изменился, создан или удален
      Console.WriteLine("File: " + e.FullPath + " " + e.ChangeType);
    }

    private static void OnRenamed(object source, RenamedEventArgs e)
    {
      // Файл переименован
      Console.WriteLine("File: {0} renamed to {1}",
                        e.OldFullPath, e.FullPath);
    }
  }
}
```

14.15. Получение короткого имени файла из длинного и наоборот

Функций .NET для выполнения этой операции нет. Можно использовать функцию Win32 (листинг 14.4).

Листинг 14.4. Получение короткого имени файла из длинного и наоборот

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

namespace ShortFileName
{
  class Class1
  {
    [DllImport("kernel32.dll", CharSet = CharSet.Auto)]
    public static extern int GetLongPathName
 (
      [MarshalAs(UnmanagedType.LPTStr)]
      string path,
      [MarshalAs(UnmanagedType.LPTStr)]
      StringBuilder longPath,
```

```
      int longPathLength
    );

    [DllImport("kernel32.dll", CharSet = CharSet.Auto)]
    public static extern int GetShortPathName
 (
      [MarshalAs(UnmanagedType.LPTStr)]
      string path,
      [MarshalAs(UnmanagedType.LPTStr)]
      StringBuilder shortPath,
      int shortPathLength
    );


    [STAThread]
    static void Main(string[] args)
    {
      // Из длинного в короткое
      StringBuilder shortPath = new StringBuilder(255);
      GetShortPathName(
        @"F:\Documents and Settings\PVA\My Documents\Книга1.xls",
        shortPath, shortPath.Capacity);
      Console.WriteLine(shortPath.ToString());

      // Из короткого в длинное
      StringBuilder longPath = new StringBuilder(255);
      GetLongPathName(shortPath.ToString(), longPath,
                                            longPath.Capacity);
      Console.WriteLine(longPath.ToString());
    }
  }
}
```

ГЛАВА 15

Сетевая информация и сетевые операции

15.1. Разбор URL на составляющие

Строку адреса можно разобрать на составляющие части (адрес, порт, протокол и т. д.), используя класс `System.UriBuilder` (листинг 15.1). С помощью этого класса можно выполнить и обратное преобразование.

Листинг 15.1. Получение короткого имени файла из длинного и наоборот

```
using System;

namespace UriParse
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      UriBuilder parser =
       new UriBuilder("http://microsoft.com:80/default.aspx?id=55");
      // microsoft.com
      Console.WriteLine(parser.Host);
      // http
      Console.WriteLine(parser.Scheme);
      // http://microsoft.com/default.aspx?id=55
      Console.WriteLine(parser.Uri);
      // /default.aspx
      Console.WriteLine(parser.Path);
```

```
      // 80
      Console.WriteLine(parser.Port);
      // ?id=55
      Console.WriteLine(parser.Query);

      UriBuilder builder = new UriBuilder(
            "https", "microsoft.com", 81, "/default.aspx","?id=77");
      // Получим https://microsoft.com:81/default.aspx?id=77
      Console.WriteLine(builder.ToString());
    }
  }
}
```

15.2. Получение DNS-имени компьютера

Метод `GetHostByName` класса `Net.Dns` позволяет получить DNS-имя компьютера:

```
string DNS = System.Net.Dns.GetHostByName("LocalHost").HostName;
```

Если имя определить не удается, метод генерирует исключение `SocketException`.

15.3. Получение NetBios-имени компьютера

Переменная `System.Environment.MachineName` содержит имя компьютера.

15.4. Получение имени хоста

Метод `System.Net.Dns.GetHostName()` возвращает имя хоста.

15.5. Получение списка IP-адресов компьютера

Для получения списка адресов используется коллекция `AddressList`:

```
foreach (System.Net.IPAddress ip in
         System.Net.Dns.GetHostByName(System.Net.Dns.
                                      GetHostName())).AddressList)
{
  Console.WriteLine(ip.ToString());
}
```

15.6. Перечисление RAS-соединений

Листинг 15.2 показывает получение списка RAS-соединений.

Листинг 15.2. Перечисление RAS-соединений

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

namespace RasEnumConnections
{
  class Class1
  {
    const int INTERNET_RAS_INSTALLED = 0x10;

    [DllImport("WININET", CharSet=CharSet.Auto)]
    static extern bool InternetGetConnectedState
 (
      ref int lpdwFlags,
      int dwReserved);

    const int MAX_PATH           = 260;
    const int RAS_MaxDeviceType  = 16;
    const int RAS_MaxPhoneNumber = 128;
    const int RAS_MaxEntryName   = 256;
    const int RAS_MaxDeviceName  = 128;

    const int RAS_Connected = 0x2000;

    [DllImport("RASAPI32", SetLastError=true, CharSet=CharSet.Auto)]
    static extern int RasEnumConnections(
      [In, Out] RASCONN[] lprasconn,
      ref int lpcb,
      ref int lpcConnections);

    [DllImport("RASAPI32", SetLastError=true, CharSet=CharSet.Auto)]
    static extern int RasGetConnectStatus(
      IntPtr hrasconn,
      ref RASCONNSTATUS lprasconnstatus);

    [StructLayout(LayoutKind.Sequential,CharSet=CharSet.Auto)]
      struct RASCONN
```

```
{
  public int  dwSize;
  public IntPtr hrasconn;
  [MarshalAs(UnmanagedType.ByValTStr,
             SizeConst=RAS_MaxEntryName+1)]
  public string szEntryName;
  [MarshalAs(UnmanagedType.ByValTStr,
             SizeConst=RAS_MaxDeviceType+1)]
  public string szDeviceType;
  [MarshalAs(UnmanagedType.ByValTStr,
             SizeConst=RAS_MaxDeviceName+1)]
  public string szDeviceName;
  [MarshalAs(UnmanagedType.ByValTStr,SizeConst=MAX_PATH)]
  public string szPhonebook;
  public int   dwSubEntry;
}

[StructLayout(LayoutKind.Sequential,CharSet=CharSet.Auto)]
  struct RASCONNSTATUS
{
  public int dwSize;
  public int rasconnstate;
  public int dwError;
  [MarshalAs(UnmanagedType.ByValTStr,
             SizeConst=RAS_MaxDeviceType+1)]
  public string szDeviceType;
  [MarshalAs(UnmanagedType.ByValTStr,
             SizeConst=RAS_MaxDeviceName+1)]
  public string szDeviceName;
  [MarshalAs(UnmanagedType.ByValTStr,
             SizeConst=RAS_MaxPhoneNumber+1)]
  public string szPhoneNumber;
}

[STAThread]
static void Main(string[] args)
{
  // Проверка, что RAS установлен
  int flags = 0;
  InternetGetConnectedState(ref flags, 0);
  if (! ((flags & INTERNET_RAS_INSTALLED) ==
                                      INTERNET_RAS_INSTALLED) )
    throw new NotSupportedException();
```

```
      // Буферы для вызова функций API
      int ret;
      int conns = 0;
      RASCONN[] rarr = new RASCONN[256];
      rarr.Initialize();
      rarr[0].dwSize = Marshal.SizeOf(typeof(RASCONN));
      int lr = rarr[0].dwSize * rarr.Length;

      // Вызываем RasEnumConnections
      ret = RasEnumConnections(rarr, ref lr, ref conns);
      if (ret != 0) throw new Win32Exception(ret);

      // Обход всех соединений
      for(int i=0;i<conns;i++)
      {
        // Очередной элемент
        RASCONN r = rarr[i];

        // Ошибочное соединение
        if (r.hrasconn == IntPtr.Zero) continue;

        // Статус соединения
        RASCONNSTATUS rcs = new RASCONNSTATUS();
        rcs.dwSize = Marshal.SizeOf(typeof(RASCONNSTATUS));

        ret = RasGetConnectStatus(r.hrasconn, ref rcs);
        if (ret != 0) throw new Win32Exception(ret);

        // Выводим информацию о соединении
        Console.WriteLine("RAS CONNECTION {0}:", i+1);
        Console.WriteLine("\tDevice Name : {0}", r.szDeviceName);
        Console.WriteLine("\tDevice Type : {0}", r.szDeviceType);
        Console.WriteLine("\tPhone Number: {0}", rcs.szPhoneNumber);
        Console.WriteLine("\tConnected? : {0}",
                          rcs.rasconnstate == RAS_Connected);
        Console.WriteLine("\tEntry Name : {0}", r.szEntryName);
        Console.WriteLine("\tPhone Book : {0}", r.szPhonebook);
        Console.WriteLine();
      }
    }
  }
}
```

15.7. Отправка письма через SMTP

Для Windows 2000 можно использовать класс `System.Web.Mail.SmtpMail`. Для его применения требуется подключение библиотеки `System.Web.Mail`. Само сообщение представляет собой экземпляр класса `System.Web.Mail.MailMessage` (листинг 15.3).

Листинг 15.3. Отправка письма через SMTP

```
using System;
using System.Web.Mail;

namespace SMTP_Send
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      SmtpMail.SmtpServer = "smtp.mail.ru";

      MailMessage mailer  = new MailMessage();
      mailer.From    = "user@mail.ru";
      mailer.Body    = "текст письма";
      mailer.Subject = "заголовок письма";
      mailer.To      = "=== кому ===";
      // mailer.Bcc =
      // mailer.Cc  =

      try
      {
        SmtpMail.Send(mailer);
      }
      catch (Exception e)
      {
        throw new Exception(e.Message);
      }

    }
  }
}
```

Следует помнить, что этот код будет работать только в случае, когда адрес `from` имеет тот же домен, что и почтовый сервер, т. е. если сервер `smtp.mail.ru`, то отправка возможна только с адресов `??@mail.ru`.

15.7.1. Отправка письма через SMTP с авторизацией

Для авторизации используются параметры, задаваемые с помощью поля `Fields` (листинг 15.4).

Листинг 15.4. Отправка письма через SMTP с авторизацией

```
using System;
using System.Web.Mail;

namespace SMTP_Test_Aut
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      SmtpMail.SmtpServer = "smtp.mail.ru";

      MailMessage mailer  = new MailMessage();

      // Авторизация SMTP
      mailer.Fields["http://schemas.microsoft.com/cdo/configuration
          /smtpauthenticate"] = 1; // cdoBasic
      mailer.Fields["http://schemas.microsoft.com/cdo/configuration
          /sendusername"    ] = "user";
      mailer.Fields["http://schemas.microsoft.com/cdo/configuration
          /sendpassword"    ] = "password";
      mailer.Fields["http://schemas.microsoft.com/cdo/configuration
          /sendusing"       ] = 2; // cdoSendUsingPort
      mailer.From    = "user@mail.ru";
      mailer.Body    = "текст письма";
      mailer.Subject = "заголовок письма";
      mailer.To      = "=== кому ===";
      // mailer.Bcc =
      // mailer.Cc  =

      try
      {
        SmtpMail.Send(mailer);
      }
      catch (Exception e)
      {
```

```
        throw new Exception(e.Message);
      }
    }
  }
}
```

15.7.2. Отправка письма через Pickup-каталог

Один из простейших способов посылки письма без помощи дополнительных клиентов — использование SMTP, встроенного в стандартную поставку Windows 2000/Server 2003. Все, что надо сделать, — создать текстовый файл в формате стандартного почтового сообщения (см. RFC 822) и положить этот файл в Pickup-каталог на SMTP-сервере. Как только файл попадет в этот каталог, он будет отправлен адресату. Обычно этот каталог располагается по адресу C:\Inetpub\Mailroot.

Для указания, что надо использовать стандартную отправку, следует задать следующие свойства (см. листинг 15.1).

```
Fields["http://schemas.microsoft.com/cdo/
       configuration/sendusing"] = 1;
Fields["http://schemas.microsoft.com/cdo/
      configuration/smtpserverpickupdirectory"]=
                    "C:\\Inetpub\\mailroot\\Pickup";
```

Кроме того, можно отключить буферизацию, установив свойство [`http://schemas.microsoft.com/cdo/configuration/flushbuffersonwrite`] в значение `true`. Это несколько снизит скорость работы, но зато увеличит надежность — каждое сообщение будет записано сразу на диск.

15.7.3. Задание кодировки сообщения

Поле [`http://schemas.microsoft.com/cdo/configuration/languagecode`] задает кодировку (язык) сообщения. Значение задается строкой, например, `ru` или `en-us`.

15.7.4. Отправка письма без использования *System.Web.Mail*

Отправка письма возможна и без использования встроенных mail-классов. Листинг 15.5 демонстрирует отправку письма напрямую через сокеты (**www.aspemporium.com/howto.aspx?hid=25**).

Листинг 15.5. Отправка почты с помощью сокетов

```
using System;
using System.IO;
```

```
using System.Net.Sockets;
using System.Net;
using System.Text;

namespace SendMail_Socket
{
  class Class1
  {
    static void Main()
    {
      string subject    = "Test mail";
      string to_name    = "Pasha";
      string to_mail    = "<???@???>";
      string from_name  = "PVA";
      string from_mail  = "<???@???>";
      string smtpServer = "smtp.mail.ru";
      int     smtpPort  = 25;

      // generate an RFC compliant email

      // Заголовок
      StringBuilder sb = new StringBuilder();
      sb.AppendFormat("Subject: {0}", subject);
      sb.Append(Environment.NewLine);
      sb.AppendFormat("To: {0}{1}", to_name, to_mail);
      sb.Append(Environment.NewLine);
      sb.AppendFormat("From: {0}{1}", from_name, from_mail);
      sb.Append(Environment.NewLine);
      sb.AppendFormat("Date: {0}",
               DateTime.Now.ToString("ddd, d MMM yyyy H:m:s zz00"));
      sb.Append(Environment.NewLine);
      sb.AppendFormat("Content-Type: text/plain");
      sb.Append(Environment.NewLine);
      sb.AppendFormat("X-Mailer: C# Mailer Example
                      (http://www.aspemporium.com/)");
      sb.Append(Environment.NewLine);
      // Тело письма
      sb.Append("Test");
      sb.Append(Environment.NewLine);

      string email = sb.ToString();

      // Отправка письма с использованием сокета

      TcpClient client = null;
```

```
try
{
  // Установка соединения
  client = new TcpClient(smtpServer, smtpPort);
  NetworkStream ns = client.GetStream();
  StreamReader stdIn  = new StreamReader(ns);
  StreamWriter stdOut = new StreamWriter(ns);

  // Ждем ответа сервера
  int responseCode = GetResponse(stdIn);
  if (responseCode != 220)
    throw new Exception("no smtp server at
           specified address or smtp server not ready");

  // Посылаем команду HELO
  stdOut.WriteLine("HELO " + Dns.GetHostName());
  stdOut.Flush();
  responseCode = GetResponse(stdIn);
  if (responseCode != 250)
    throw new Exception("helo fails. code="+responseCode);

  // Команда MAIL
  stdOut.WriteLine("MAIL FROM:"+from_mail);
  stdOut.Flush();
  responseCode = GetResponse(stdIn);
  if (responseCode != 250)
    throw new Exception("FROM email considered
                  bad by server. code="+responseCode);

  // Команда RCPT
  stdOut.WriteLine("RCPT TO:"+to_mail);
  stdOut.Flush();
  responseCode = GetResponse(stdIn);
  switch(responseCode)
  {
    case 250:
    case 251:
      break;
    default:
      throw new Exception("TO email considered
                    bad by server. code="+responseCode);
  }
```

```
      // Команда DATA
      stdOut.WriteLine("DATA");
      stdOut.Flush();
      responseCode = GetResponse(stdIn);
      if (responseCode != 354)
        throw new Exception("data command
                        not accepted. code="+responseCode);

      // Отправка
      stdOut.WriteLine(email);
      stdOut.Flush();

      // Отправка одиночной точки означает завершение отправки
      stdOut.WriteLine(".");
      stdOut.Flush();
      responseCode = GetResponse(stdIn);
      if (responseCode != 250)
        throw new Exception("email not
                        accepted. code="+responseCode);

      // Команда QUIT
      stdOut.WriteLine("QUIT");
      stdOut.Flush();
      responseCode = GetResponse(stdIn);
      if (responseCode != 221)
      {
        // who cares
      }

      Console.WriteLine("Все успешно отправлено");

    }
    catch(Exception ex)
    {
      Console.WriteLine("Ошибка: " + ex.Message);
    }
    finally
    {
      // Закрываем соединение
      if (client != null)
        client.Close();
      client = null;
    }
  }
```

```
    static int GetResponse(StreamReader stdIn)
    {
      try
      {
        string response = string.Empty;
        // Читаем ответ сервера
        do
        {
          response += stdIn.ReadLine()+"\r\n";
        }
        while(stdIn.Peek() != -1);

        // Выводим на консоль
        Console.WriteLine(response);

        // Получаем код ответа (первые три символа)
        return Convert.ToInt32(response.Substring(0, 3));
      }
      catch
      {
        // Если ошибка
        return 0;
      }
    }
  }
}
```

15.8. Получение списка сетевых дисков

Перечисление сетевых дисков (листинг 15.6) производится с помощью библиотеки `IWshRuntimeLibrary` *(см. разд. 12.7)*.

Листинг 15.6. Перечисление сетевых дисков

```
using System;
using System.Collections;
using IWshRuntimeLibrary;

namespace EnumNetworkDrives
{
  class Class1
  {
    [STAThread]
```

```
    static void Main(string[] args)
    {
      WshNetwork network = new WshNetwork();
      foreach (IEnumerable driver in network.EnumNetworkDrives())
      {
        Console.WriteLine(driver.ToString());
      }
    }
  }
}
```

15.9. Получение имени текущего пользователя

Переменные `UserDomainName` и `UserName` класса `Environment` возвращают информацию о текущем пользователе системы:

```
Console.WriteLine(Environment.UserDomainName + @"\" +
                  Environment.UserName);
```

Эту же информацию можно получить с помощью класса `WindowsIdentity`:

```
WindowsIdentity user = WindowsIdentity.GetCurrent();
Console.WriteLine(user.Name.ToString());
```

15.10. Программная имперсонация

Программную имперсонацию пользователя с заданным именем и паролем можно выполнить с помощью функций, импортированных из Win32 (листинг 15.7).

Листинг 15.7. Имперсонация пользователя

```
using System;
using System.Runtime.InteropServices;
using System.Security.Principal;

namespace LogonUser
{
  class Class1
  {
    private const int LOGON32_LOGON_INTERACTIVE       = 2;
    private const int LOGON32_LOGON_NETWORK_CLEARTEXT = 3;
    private const int LOGON32_PROVIDER_DEFAULT        = 0;
```

```
[DllImport("advapi32.dll", CharSet=CharSet.Auto)]
static extern int LogonUser (string lpszUserName,
        string lpszDomain, string lpszPassword, int dwLogonType,
        int dwLogonProvider, ref IntPtr phToken);

[DllImport("advapi32.dll", CharSet=CharSet.Auto,
            SetLastError=true)]
static extern int DuplicateToken (IntPtr hToken,
        int impersonationLevel, ref IntPtr hNewToken);

[STAThread]
static void Main(string[] args)
{
  // Печатаем текущего пользователя
  WindowsIdentity wi = WindowsIdentity.GetCurrent();
  Console.WriteLine("Name={0} --> {1}", wi.Name,
                                        wi.IsAuthenticated);

  string UserName = "Administrator";
  string Password = "123";

  // Для сохранения текущей имперсонации
  WindowsImpersonationContext impersonationContext = null;

  try
  {
    // Имперсонируем другого пользователя
    WindowsIdentity newIdentity;
    IntPtr token = IntPtr.Zero;
    IntPtr tokenDuplicate = IntPtr.Zero;
    if(LogonUser(UserName, Environment.MachineName, Password,
      LOGON32_LOGON_NETWORK_CLEARTEXT,
                    LOGON32_PROVIDER_DEFAULT, ref token) != 0)
    {
      if(DuplicateToken(token, 2, ref tokenDuplicate) != 0)
      {
        newIdentity = new WindowsIdentity(tokenDuplicate);
        // При имперсонации возвращается текущее значение
        impersonationContext = newIdentity.Impersonate();
      }
    }
  }
```

```
      catch(Exception Ex)
      {
        Console.WriteLine("Ошибка имперсонации
                          пользователя {0}: {1}", UserName, Ex);
      }

      // Новая имперсонация
      WindowsIdentity wi1 = WindowsIdentity.GetCurrent();
      Console.WriteLine("Name={0} --> {1}",
                            wi1.Name, wi1.IsAuthenticated);

      // Возвращаем предыдущую имперсонацию
      if (impersonationContext != null)
        impersonationContext.Undo();

      // Печатаем снова
      WindowsIdentity wi2 = WindowsIdentity.GetCurrent();
      Console.WriteLine("Name={0} --> {1}",
                          wi2.Name, wi2.IsAuthenticated);
    }
  }
}
```

15.11. Как получить список групп домена, в которые входит пользователь?

Для этого есть три пути.

Во-первых, можно получить маркер доступа, связанный с пользователем (например, воспользоваться функцией `LogonUser`), а затем передать его в функцию `GetTokenInformation`, указав во втором параметре значение `TokenGroups`. Таким образом, можно получить список групп. Недостаток этого метода очевиден — обращение к `unmanaged`-коду.

Во-вторых, можно воспользоваться `DirectoryServices`. Для этого надо иметь право запрашивать каталог, а также знать домен пользователя и некоторые другие параметры.

Но для тех, кто не любит возиться с `unmanaged`-кодом и хочет все быстро и просто, существует третий путь. На самом деле, класс `WindowsIdentity` имеет `internal`-метод `GetRoles()`. Хотя метод и "внутренний", вызвать его можно, используя механизм отражения (reflection):

```
public static string[] GetRoles(
           System.Security.Principal.WindowsPrincipal principal)
```

```
{
   string[] roles = (string[]) CallPrivateMethod(
                principal.Identity, "GetRoles");
   return roles;
}
private static object CallPrivateMethod( object o,
                        string methodName )
{
   Type t = o.GetType();
   MethodInfo mi = t.GetMethod( methodName,
           BindingFlags.NonPublic | BindingFlags.Instance);
   if (mi == null)
   {
     throw new ReflectionTypeLoadException(null,null,
           String.Format( "{0}.{1} method wasn't found. The runtime
           implementation may have changed!",
           t.FullName, methodName ) );
   }
   return mi.Invoke(o, null);
}
```

Этот путь не рекомендуется использовать в "серьезных" приложениях, т. к. в будущих версиях Framework метод `GetRoles()` может измениться (или исчезнуть вообще). Однако для быстрого создания прототипа эта техника вполне годится.

И все-таки, где это возможно, лучше все же обойтись методом `IsInRole`. Практика показывает, что в большинстве случаев его действительно достаточно.

15.12. Получение файла из Интернета

С помощью класса `System.Net.WebClient` можно скачать файл или страницу по заданному URL (листинг 15.8).

Листинг 15.8. Получение файла из Интернета

```
using System;

namespace DownloadFile
{
  class Class1
  {
    [STAThread]
```

```
    static void Main(string[] args)
    {
      try
      {
        System.Net.WebClient client = new System.Net.WebClient();

        string url =
            @"http://i2.microsoft.com/h/en-us/i/msnlogo.gif";
        string loc = @"E:\test.jpg";
        client.DownloadFile(url, loc);
      }
      catch(Exception e)
      {
        Console.WriteLine(e.ToString());
      };
    }
  }
}
```

15.13. Получение данных из Интернета

Аналогично коду, приведенному в листинге 15.5, можно получить не только файл, но и любые данные, представленные массивом байтов. Для этого вызов `DownloadFile` надо заменить на вызов

```
byte[] data = client.DownloadData(url);
```

15.14. Получение Web-страницы

Получить Web-страницу можно с помощью класса `System.Net.WebRequest`:

```
System.Net.WebRequest req =
     System.Net.WebRequest.Create(@"http://www.microsoft.com");
System.Net.WebResponse resp = req.GetResponse();
System.IO.Stream stream = resp.GetResponseStream();
System.IO.StreamReader sr = new System.IO.StreamReader(stream);
string s = sr.ReadToEnd();
Console.WriteLine(s);
```

Или по-другому (обратите внимание на указание кодировки страницы):

```
using System.Net;
using System.IO;
using System.Text;
```

```
string url = "http://www.microsoft.com";
WebRequest req = WebRequest.Create(url);

using (WebResponse resp = req.GetResponse())
{
  using (StreamReader reader = new StreamReader(
              resp.GetResponseStream(), Encoding.UTF8))
  {
    string content = reader.ReadToEnd();
  }
}
```

15.15. Использование прокси-сервера

Задать адрес прокси-сервера можно двумя способами: глобально

```
System.Net.GlobalProxySelection.Select =
                    new WebProxy("proxyname", 80);
```

или для конкретного запроса

```
HttpWebRequest request =
    (HttpWebRequest)WebRequest.Create( "http://localhost");
request.Proxy = new WebProxy("proxyname", 80);
```

15.16. Подключен ли компьютер к Интернету?

Проверить наличие подключения к Интернету можно с помощью функций WININET (листинг 15.9).

Листинг 15.9. Проверка подключения к Интернету

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

namespace InternetAvailable
{
  class InernetChecker
  {
    [DllImport("WININET", CharSet=CharSet.Auto)]
    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
    }

    static void Main()
    {
      InternetConnectionState flags = 0;

      Console.WriteLine(
        "InternetGetConnectedState : {0} - {1}",
        (InternetGetConnectedState(ref flags, 0)?
                                          "ONLINE":"OFFLINE"),
        flags
        );
    }
  }
}
```

15.17. Как проверить, доступен ли компьютер в сети?

Проверить доступность компьютера можно с помощью функции `GetRTTAndHopCount` (листинг 15.10).

Листинг 15.10. Проверка доступности компьютера в сети

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

namespace HopCount
{
  class Class1
  {
    [DllImport("iphlpapi.dll", ExactSpelling=true,
                                          SetLastError=true)]
```

```
    private static extern bool GetRTTAndHopCount (uint DestIpAddress,
                    out int HopCount, int MaxHops, out int RTT );

    private static int GetHopCount(string pIpAddresseStr)
    {
      IPAddress lIpAddressEntry;
      byte[] lIpAddressArray;
      uint lIpAddress;

      int lTTL;
      int lHopCount;
      int lErrorCode;

      lIpAddressEntry = IPAddress.Parse(pIpAddresseStr);
      lIpAddressArray = lIpAddressEntry.GetAddressBytes();
      lIpAddress = 0;
      for(int i = 0; i < lIpAddressArray.Length; i++)
      {
        lIpAddress += (uint)(lIpAddressArray[i] << (8 * i));
      }
      if (!GetRTTAndHopCount(lIpAddress, out lHopCount,
                                              60, out lTTL))
      {
        lErrorCode = Marshal.GetLastWin32Error();
        throw new Exception("Ошибка. Код " + lErrorCode);
      }
      return lHopCount;
    }

    [STAThread]
    static void Main(string[] args)
    {
      try
      {
        Console.WriteLine(GetHopCount("127.0.0.1"));
      }
      catch (Exception e)
      {
        Console.WriteLine(e.Message);
      }
    }
  }
}
```

15.18. Как сделать ping?

Листинг 15.11 показывает программный ping (C# 2.0). Для C# 1.1 придется использовать более сложный код, реализованный на сокетах (**www.csharphelp.com/archives/archive6.html**).

Листинг 15.11. Программный ping (C# 2.0)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net.NetworkInformation;

namespace Ping2
{
  class Program
  {
    static void Main(string[] args)
    {
      Ping pingSender = new Ping();

      PingOptions options = new PingOptions();
      options.DontFragment = true;

      // Буфер 32 байта
      string data = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
      byte[] buffer = Encoding.ASCII.GetBytes(data);
      int timeout = 120;
      PingReply reply = pingSender.Send(
                "rsdn.ru", timeout, buffer, options);
      if (reply.Status == IPStatus.Success)
      {
        Console.WriteLine("Address: {0}", reply.Address.ToString());
        Console.WriteLine("RoundTrip time: {0}",
                          reply.RoundtripTime);
        Console.WriteLine("Time to live: {0}", reply.Options.Ttl);
        Console.WriteLine("Don't fragment: {0}",
                          reply.Options.DontFragment);
        Console.WriteLine("Buffer size: {0}", reply.Buffer.Length);
      }
    }
  }
}
```

15.19. Получение файла с FTP через WinInet-функции

.NET 1.1 не содержит функций для работы с FTP. Получение файла можно организовать с помощью WinInet-функций (листинг 15.12).

Листинг 15.12. Получение файла с FTP

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

namespace FTP_File
{
  internal class __FtpDownload {
    [DllImport("WININET", EntryPoint="InternetOpen",
      SetLastError=true, CharSet=CharSet.Auto)]
    static extern IntPtr __InternetOpen (
      string lpszAgent,
      int dwAccessType,
      string lpszProxyName,
      string lpszProxyBypass,
      int dwFlags);

    [DllImport("WININET", EntryPoint="InternetCloseHandle",
      SetLastError=true, CharSet=CharSet.Auto)]
    static extern bool __InternetCloseHandle (IntPtr hInternet);

    [DllImport("WININET", EntryPoint="InternetConnect",
      SetLastError=true, CharSet=CharSet.Auto)]
    static extern IntPtr __InternetConnect (
      IntPtr hInternet,
      string lpszServerName,
      int nServerPort,
      string lpszUsername,
      string lpszPassword,
      int dwService,
      int dwFlags,
      int dwContext);

    [DllImport("WININET", EntryPoint="FtpSetCurrentDirectory",
      SetLastError=true, CharSet=CharSet.Auto)]
```

```
static extern bool __FtpSetCurrentDirectory (
  IntPtr hConnect,
  string lpszDirectory);

[DllImport("WININET", EntryPoint="InternetAttemptConnect",
  SetLastError=true, CharSet=CharSet.Auto)]
static extern int __InternetAttemptConnect (int dwReserved);

[DllImport("WININET", EntryPoint="FtpGetFile",
  SetLastError=true, CharSet=CharSet.Auto)]
static extern bool __FtpGetFile (
  IntPtr hConnect,
  string lpszRemoteFile,
  string lpszNewFile,
  bool fFailIfExists,
  FileAttributes dwFlagsAndAttributes,
  int dwFlags,
  int dwContext);

const int ERROR_SUCCESS             = 0;
const int INTERNET_OPEN_TYPE_DIRECT = 1;
const int INTERNET_SERVICE_FTP      = 1;

static void Main()
{
  IntPtr inetHandle = IntPtr.Zero;
  IntPtr ftpconnectHandle = IntPtr.Zero;

  try
  {
    // Проверяем соединение с Интернетом
    if (__InternetAttemptConnect(0) != ERROR_SUCCESS)
    {
      throw new InvalidOperationException(
        "Нет соединения с Интернетом");
    }

    // Подключаемся к Интернету
    inetHandle = __InternetOpen(
      "billyboy FTP", INTERNET_OPEN_TYPE_DIRECT, null, null, 0);
    if (inetHandle == IntPtr.Zero)
    {
      throw new NullReferenceException(
        "Не удалось подключиться к Интернету");
    }
```

```
    // Подключаемся к ftp.microsoft.com
    ftpconnectHandle = __InternetConnect(
      inetHandle, "ftp.microsoft.com", 21, "anonymous",
      "myemail@yahoo.com", INTERNET_SERVICE_FTP,
      0, 0);
    if (ftpconnectHandle == IntPtr.Zero)
    {
      throw new NullReferenceException(
        "Не удалось подключиться к ftp.microsoft.com");
    }

    // Переходим в нужный каталог на FTP
    if (! __FtpSetCurrentDirectory(ftpconnectHandle,
                                   "/deskapps"))
    {
      throw new InvalidOperationException(
        "Не удалось сменить каталог");
    }

    // Загружаем файл с сервера
    if (! __FtpGetFile(ftpconnectHandle, "readme.txt",
                       "c:\\downloadedFile1.txt",
                       false, 0, 0, 0))
    {
      throw new IOException("Не удалось загрузить файл");
    }

    // Файл успешно загружен
    Console.WriteLine("SUCCESS: файл загружен успешно");
  } catch (Exception ex) {
    // Выводим сообщение, если ошибка
    Console.WriteLine("ERROR: " + ex.Message);
  } finally {
    // Закрываем соединение с ftp.microsoft.com
    if (ftpconnectHandle != IntPtr.Zero) {
      __InternetCloseHandle(ftpconnectHandle);
    }
    ftpconnectHandle = IntPtr.Zero;
    // Закрываем соединение с Интернетом
    if (inetHandle != IntPtr.Zero) {
      __InternetCloseHandle(inetHandle);
    }
```

```
        inetHandle = IntPtr.Zero;
      }
    }
  }
}
```

15.20. Перечисление компьютеров в сети

Для перечисления компьютеров в сети можно использовать функцию `NetServerEnum` (листинг 15.13).

Листинг 15.13. Перечисление компьютеров в сети

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

namespace ServerEnum
{
  internal class Class1
  {
    [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);

    //
    // типы серверов
    //
    [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,
}

[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;
}

//
// Операционная система
//
public enum PLATFORM_ID : uint
```

```
{
  PLATFORM_ID_DOS = 300,
  PLATFORM_ID_OS2 = 400,
  PLATFORM_ID_NT  = 500,
  PLATFORM_ID_OSF = 600,
  PLATFORM_ID_VMS = 700,
}

//
// 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 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, type, 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);
    }

    [STAThread]
    static void Main()
    {
      Console.WriteLine("{0} WORKSTATION", new string('=', 10));
      ArrayList list =
                GetServerList(SV_101_TYPES.SV_TYPE_WORKSTATION);
      foreach (string name in list)
      {
        Console.WriteLine(name);
      }
      Console.WriteLine("{0} UNIX", new string('=', 10));
      list = GetServerList(SV_101_TYPES.SV_TYPE_SERVER_UNIX);
      foreach (string name in list)
      {
        Console.WriteLine(name);
      }
    }
  }
}
```

ГЛАВА 16

Рабочий стол и панель управления

16.1. Что такое *IWshRuntimeLibrary* и *Shell32*?

Многие перечисленные далее операции выполняются с помощью классов библиотеки `IWshRuntimeLibrary`. Для ее использования следует подключить COM-объект **Windows Script Host Object Model** (модуль wshom.ocx, рис. 16.1).

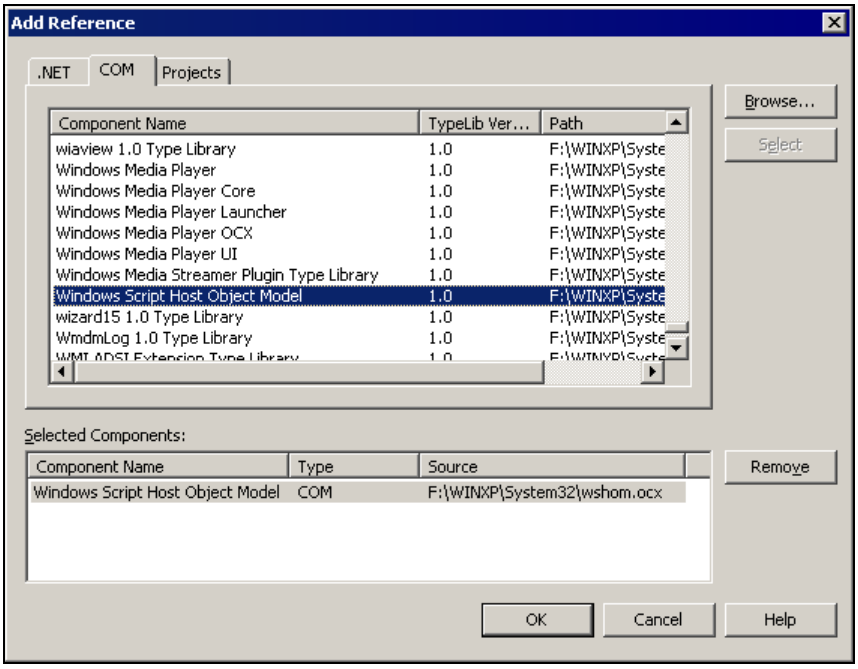

Рис. 16.1. Подключение библиотеки **Windows Script Host Object Model**

Другие операции выполняются с помощью классов библиотеки `Shell32`, для использования которой следует подключить COM-объект **Microsoft Shell Controls And Automation** (модуль shell32.Dll).

16.2. Создание ярлыка

Создание ярлыка (например, на рабочем столе) производится с помощью библиотеки `IWshRuntimeLibrary` (листинг 16.1).

Листинг 16.1. Создание ярлыка на рабочем столе

```
using System;
using IWshRuntimeLibrary;

namespace AddShortcutToDesktop
{
  class ShortcutToDesktop
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Создать ярлык на рабочем столе
      object shortAdr = (object)"Desktop";

      WshShell shell = new WshShell();
      // Получить полный адрес ярлыка
      string link = ((string) shell.SpecialFolders.Item(
                           ref shortAdr)) + @"\Calc.lnk";
      // Создать объект ярлыка
      IWshShortcut shortcut =
                         (IWshShortcut)shell.CreateShortcut(link);
      // Описание
      shortcut.Description = "Ярлык для калькулятора";
      // Установить горячую клавишу
      shortcut.Hotkey = "CTRL+SHIFT+A";
      // Получить путь для программы Калькулятор
      shortcut.TargetPath =
       Environment.GetFolderPath(Environment.SpecialFolder.System) +
       @"\Calc.exe";
      // Создать ярлык
      shortcut.Save();
    }
  }
}
```

Кроме имени **Desktop** (см. переменную `shortAdr`) можно использовать имена, перечисленные в табл. 16.1.

Таблица 16.1. *Имена для метода* `GetFolderPath`

Имя	Описание
`AllUsersDesktop`	Рабочий стол для всех пользователей
`AllUsersStartMenu`	Меню **Пуск**, доступное для всех пользователей
`AllUsersPrograms`	Ярлыки, доступные для всех пользователей
`AllUsersStartup`	Меню автозапуска для всех пользователей
`Desktop`	Рабочий стол текущего пользователя
`Favorites`	Папка Избранное (Internet favorite) для текущего пользователя
`Fonts`	Папка системных шрифтов
`MyDocuments`	Папка Мои документы для текущего пользователя
`NetHood`	Папка сетевых ярлыков для текущего пользователя
`PrintHood`	Папка ярлыков принтеров для текущего пользователя
`Programs`	Папка программ текущего пользователя
`Recent`	Ярлыки последних открытых документов текущего пользователя
`SendTo`	Ярлыки меню **Отправить** (Send To), доступного по щелчку правой кнопки мыши для любого файла
`StartMenu`	Меню **Пуск** текущего пользователя
`Startup`	Меню автозапуска программ для текущего пользователя
`Templates`	Папка шаблонов для текущего пользователя

16.3. Как найти путь к папке автозапуска?

Путь к папке автозапуска можно найти с помощью вызова метода `GetFolderPath`:

```
System.Environment.GetFolderPath(
          System.Environment.SpecialFolder.Startup);
```

16.4. Регистрация меню запуска программы по расширению

Зарегистрировать меню для файлов с определенным расширением можно с помощью кода, представленного в листинге 16.2. Следует иметь в виду, что

каждый запуск такого кода добавляет новую строку в меню (рис. 16.2). В реальном коде необходимо сначала проверить, не создавался ли такой ключ ранее.

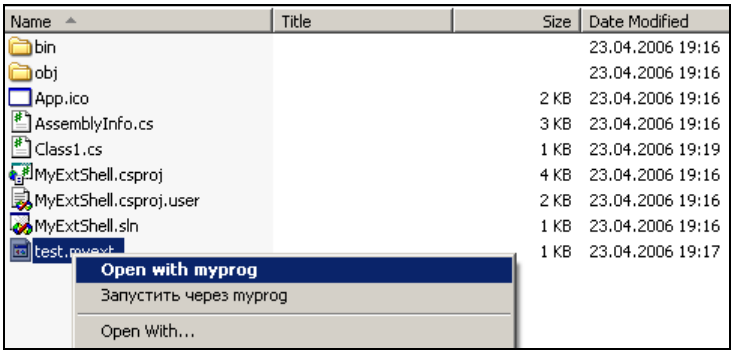

Рис. 16.2. Регистрация пункта меню

Листинг 16.2. Регистрация меню

```
using System;
using Microsoft.Win32;

namespace MyExtShell
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      RegistryKey newkey;
      newkey = Registry.ClassesRoot.CreateSubKey("myprog_file");
      newkey = newkey.CreateSubKey("shell");
      newkey = newkey.CreateSubKey("Запустить через myprog");
      newkey = newkey.CreateSubKey("command");
      newkey.SetValue ("", @"c:\myfolder\myprog.exe %1");
      newkey = Registry.ClassesRoot.CreateSubKey(".myext");
      newkey.SetValue ("", "myprog_file");
    }
  }
}
```

16.5. Доступ к элементам панели управления

Доступ к элементам панели управления осуществляется с помощью библиотеки `Shell32` (листинг 16.3).

Листинг 16.3. Запуск элементов панели управления

```
using System;
using Shell32;

namespace ShellDemo
{
  class ShellDemoClass
  {
    [STAThread]
    static void Main(string[] args)
    {
      Shell shell = new Shell();

      // Доступ к компонентам панели управления
      shell.ControlPanelItem("desk.cpl");
    }
  }
}
```

Кроме имени desk.cpl (см. вызов `ControlPanelItem`) можно использовать имена компонентов панели управления, некоторые из которых перечислены в табл. 16.2.

Таблица 16.2. *Имена cpl-модулей*

Компонент	Имя
Установка и удаление программ (Add/remove programs)	appwiz.cpl
Рабочий стол (Desktop settings)	desk.cpl
Настройка DirectX (DirectX properties)	directx.cpl
Установка оборудования (Add hardware wizard)	hdwwiz.cpl
Свойства обозревателя (Internet properties)	inetcpl.cpl
Язык и стандарты (regional and language options)	intl.cpl
Игровые устройства (Game controllers)	joy.cpl
Мышь (Mouse properties)	main.cpl

Таблица 16.2 *(окончание)*

Компонент	Имя
Звук (Sounds and audio devices properties)	mmsys.cpl
Сеть (Network connections)	ncpa.cpl
Пользователи (User accounts)	nusrmgr.cpl
Система (System properties)	sysdm.cpl
Дата и время (Date and time properties)	timedate.cpl
Windows Update (Automatic updates — WindowsUpdate settings)	wuaucpl.cpl

16.6. Как получить разрешение экрана?

Для получения разрешения экрана нужно использовать свойство `System.Windows.Forms.Screen.PrimaryScreen.Bounds`.

ГЛАВА 17

Потоки

17.1. Запуск и остановка потоков

Методы для работы с потоками находятся в пространстве имен `System.Threading` (листинг 17.1).

Листинг 17.1. Запуск потока

```
using System;
using System.Threading;

namespace Threads_Standard
{
  public class ThreadClass
  {
    // Основной метод потока
    public void Execute()
    {
    }
  }

  class Threads_Standard
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Создаем класс потока
      ThreadClass tc = new ThreadClass();
      // Создаем сам поток
      Thread t = new Thread(new ThreadStart(tc.Execute));
```

```
      // Старт потока
      t.Start();

      // Еще один поток с помощью статического метода
      Thread t1 = new Thread(new ThreadStart(Static_Execute));
      t1.Start();

      // После первого нажатия клавиши <Enter> завершим поток
      Console.ReadLine();
      t.Abort();
      t1.Abort();

      // Ждать реального завершения потока
      t.Join(1000); // в течение 1 секунды
      t1.Join();    // просто ждать

      // Просто выходим по нажатию клавиши <Enter>
      Console.WriteLine("Нажмите Enter для выхода...");
      Console.ReadLine();
    }

    // Статический метод
    static void Static_Execute()
    {
    }
  }
}
```

Завершение потока можно выполнять двумя способами: с помощью метода `Abort()` и с помощью метода `Interrupt()`. Первый метод вызывает в потоке исключение `ThreadAbortException`, и поток завершается. Второй метод вызывает исключение `ThreadInterruptedException`, но поток может завершить работу в удобный для него момент времени. Кроме того, метод `Abort` может принимать специальный параметр типа `object`, позволяющий передать информацию в поток о причине завершения.

Ожидание завершения потока производится с помощью метода `Join`.

Каждый поток имеет имя, задать которое можно с помощью свойства `Name`:

```
Thread.CurrentThread.Name = Name;
```

В C# предусматривается механизм обратного вызова методов. С его помощью вызывающий класс может передать ссылку на метод, который будет вызывать поток. Однако следует учитывать, что этот метод будет вызываться в контексте потока.

Пример использования этих методов продемонстрирован в листинге 17.2 (сам класс потока приведен в листинге 17.3).

Листинг 17.2. Запуск потока, остановка, передача callback, задание имени, передача информации через метод `Abort`

```
using System;
using System.Threading;

namespace Threads_Create
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Создаем класс потока. Один из параметров - callback-функция
      ThreadClass tc = new ThreadClass("Thread1",
                        new ThreadCallback(ResultCallback));
      // Создаем сам поток
      Thread t = new Thread(new ThreadStart(tc.Execute));
      // Старт потока
      t.Start();

      // Еще один поток
      ThreadClass tc1 = new ThreadClass("Thread2",
                        new ThreadCallback(ResultCallback));
      Thread t1 = new Thread(new ThreadStart(tc1.Execute));
      t1.Start();

      // После первого нажатия клавиши <Enter> завершим поток
      Console.ReadLine();
      // Завершаем через вызов исключения ThreadAbortException.
      // Вообще говоря, так делать не рекомендуется
      t.Abort("Стоп из main");
      // Завершаем через вызов исключения ThreadInterruptedException
      t1.Interrupt();

      // Ждем реального завершения
      t.Join();
      t1.Join();

      // Просто выходим по нажатию клавиши <Enter>
      Console.WriteLine("Нажмите Enter для выхода...");
```

```
      Console.ReadLine();
    }

    public static int cnt = 1;

    // Callback-функция для потока
    public static void ResultCallback(string Name, int Counter)
    {
      lock(Console.Out)
      {
        Console.WriteLine("Счетчик потока {0} равен {1} cnt={2}",
                          Name, Counter, cnt);
        if (Name == "Thread1") cnt++; else cnt--;
      }
    }
  }
}
```

Листинг 17.3. Класс потока для листинга 17.2

```
using System;
using System.Threading;

namespace Threads_Create
{
  // Описание callback-функции потока
  public delegate void ThreadCallback(string Name, int Counter);

  public class ThreadClass
  {
    private string Name;
    private int  Counter;
    private ThreadCallback Callback;
    private bool Stop;

    public ThreadClass(string Name, ThreadCallback Callback)
    {
      this.Name   = Name;
      this.Callback = Callback;

      Counter = 0;
      Stop = false;
    }
```

```
// Основной метод потока
public void Execute()
{
  // Имя потока
  Thread.CurrentThread.Name = Name;
  // Приоритет
  Thread.CurrentThread.Priority = ThreadPriority.Normal;

  // Основной цикл потока
  while (!Stop)
  {
    try
    {
      Console.WriteLine(string.Format("Выполнение
                                       потока {0}", Name));
      Counter++;

      // Вызов callback
      if (Callback != null)
        Callback(Name, Counter);

      // Спать заданное количество миллисекунд
      Thread.Sleep(1);
    }
    // При вызове метода Abort будет сгенерировано исключение,
    // которое мы ловим для завершения потока
    catch (ThreadAbortException ex_abort)
    {
      // Обработать исключение мы можем, но поток уже остановлен.
      // Сообщение про завершение потока не будет
      Console.WriteLine(string.Format("Завершение потока {0}
                            (ThreadAbortException)", Name));
      Console.WriteLine(string.Format("Информация: {0}",
                           (string)ex_abort.ExceptionState));
    }
    catch (ThreadInterruptedException)
    {
      Console.WriteLine(string.Format("Завершение потока {0}
                        (ThreadInterruptedException)", Name));
      // Это исключение только спрашивает
      // о возможности завершения потока,
      // мы должны сами завершить поток
      Stop = true;
    }
  }
```

```
      Console.WriteLine(string.Format("Выполнение потока {0}
                                завершено.", Name));
    }
  }
}
```

17.2. Отмена остановки потока

Метод `Thread.ResetAbort()` позволяет потоку продолжить выполнение после выброса исключения `ThreadAbortException`:

```
catch (ThreadAbortException)
{
  // Игнорируем Abort
  Thread.ResetAbort();
}
```

17.3. Изолирование данных потока

Если поток организуется с помощью статического метода, то возникает проблема обеспечения независимости данных потока. Обычная статическая переменная будет являться общей для класса и перезаписываться при последнем использовании.

Сделать статические переменные изолированными для каждого потока можно двумя способами. Первый — использовать слот (slot), второй — установить атрибут `ThreadStatic` (листинг 17.4). Сам класс потока приведен в листинге 17.5.

Листинг 17.4. Изолированные и статические данные

```
using System;
using System.Threading;
using System.IO;

namespace Threads_Slot
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      Thread t1 = new Thread(new ThreadStart(ThreadClass.Execute));
      t1.Start();
```

```
      Thread t2 = new Thread(new ThreadStart(ThreadClass.Execute));
      t2.Start();

      // После первого нажатия клавиши <Enter> завершим поток
      Console.ReadLine();
      // Завершаем
      t1.Interrupt(); t1.Join();
      t2.Interrupt(); t2.Join();

      // Просто выходим по нажатию клавиши <Enter>
      Console.WriteLine("Нажмите Enter для выхода...");
      Console.ReadLine();
    }
  }
}
```

Листинг 17.5. Класс потока для листинга 17.4

```
using System;
using System.Threading;
using System.IO;

namespace Threads_Slot
{
  public class ThreadClass
  {
    static Random randomGenerator = new Random();

    // Такой static-объект будет общим между всеми потоками
    // этого класса
    static int intShare;

    // Такой static-объект будет уникальным для потока
    [ThreadStatic]
    static int intUnic;

    // Данные из слота будут разными для всех потоков
    protected static int SlotData
    {
      get
      {
        object data =
                Thread.GetData(Thread.GetNamedDataSlot("MySlot"));
```

```
        if (data == null)
          return 0;
        return Convert.ToInt32(data);
      }
      set
      {
        Thread.SetData(Thread.GetNamedDataSlot("MySlot"), value);
      }
    }

    // Статический метод потока
    public static void Execute()
    {
      SlotData = randomGenerator.Next(1, 200);
      intShare = SlotData;
      intUnic = SlotData;

      bool Stop = false;

      while (!Stop)
      {
        try
        {
          // Хотя класс потока статический,
          // slot-данные будут иметь разное значение.
          // Обычный static будет перезаписан,
          // static с атрибутом будут иметь разные значения
          Console.WriteLine("SlotData={0} intShare={1} intUnic={2}",
                    SlotData, intShare, intUnic);
          Thread.Sleep(1000);
        }
        catch (ThreadAbortException)
        {
        }
        catch (ThreadInterruptedException)
        {
          Stop = true;
        }
      }
    }
  }
}
```

17.4. Права потоков

По умолчанию запускаемые потоки не имеют прав на ресурсы. Для установки прав необходимо вызвать метод `SetThreadPrincipal` (листинг 17.6).

Листинг 17.6. Вызов `SetThreadPrincipal`

```
using System;
using System.Security.Principal;
using System.Threading;

class Threads_Principal
{
  static void Main(string[] args)
  {
    // Поток с правами по умолчанию
    Thread t =
          new Thread(new ThreadStart(PrintPrincipalInformation));
    t.Start();
    t.Join();

    // Задание WindowsPrincipal
    AppDomain currentDomain = AppDomain.CurrentDomain;
    currentDomain.SetPrincipalPolicy(
                    PrincipalPolicy.WindowsPrincipal);

    // Создание потока с правами WindowsPrincipal
    t = new Thread(new ThreadStart(PrintPrincipalInformation));
    t.Start();
    t.Join();

    // Задание прав NewUser
    IIdentity identity = new GenericIdentity("NewUser");
    IPrincipal principal = new GenericPrincipal(identity, null);
    currentDomain.SetThreadPrincipal(principal);

    // Поток с правами пользователя NewUser
    t = new Thread(new ThreadStart(PrintPrincipalInformation));
    t.Start();
    t.Join();

    // Ждем нажатия клавиши <Enter> для завершения программы
    Console.ReadLine();
  }
```

```
  static void PrintPrincipalInformation()
  {
    IPrincipal curPrincipal = Thread.CurrentPrincipal;
    if(curPrincipal != null)
    {
      Console.WriteLine("Тип: " + curPrincipal.GetType().Name);
      Console.WriteLine("Имя: " + curPrincipal.Identity.Name);
      Console.WriteLine("Идентификация: " +
                  curPrincipal.Identity.IsAuthenticated);
      Console.WriteLine();
    }
  }
}
```

17.5. Идентификаторы потоков

Уникальный номер потока можно получить двумя способами:

```
AppDomain.GetCurrentThreadId().ToString()
Thread.CurrentThread.GetHashCode().ToString()
```

Есть еще возможность вызвать функцию `GetCurrentThreadId()` из Win32, но это будет работать довольно медленно.

17.6. Пул потоков

Использование пула потоков показано в листинге 17.7.

Листинг 17.7. Использование пула потоков

```
using System;
using System.Threading;

namespace Threads_Pool
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Функция потока
      WaitCallback callback = new WaitCallback(ThreadFunction);
```

```
      // Создаем три потока в пуле
      ThreadPool.QueueUserWorkItem(callback, "Thread1");
      ThreadPool.QueueUserWorkItem(callback, "Thread2");
      // Unsafe-поток
      ThreadPool.UnsafeQueueUserWorkItem(callback, "Thread3");

      // Просто выходим по нажатию клавиши <Enter>
      Console.WriteLine("Нажмите Enter для выхода...");
      Console.ReadLine();
    }
    static void ThreadFunction(object state)
    {
      Console.WriteLine("Поток с параметром {0}", state.ToString());
      Thread.Sleep(1000);
      Console.WriteLine("Завершение потока {0}", state.ToString());
    }
  }
}
```

17.7. Синхронизация потоков

Для предотвращения одновременного доступа к ресурсам используется класс `System.Threading.Monitor` (листинг 17.8). В C# существует специальный оператор `lock`, который позволяет записать этот код в более компактном виде (листинг 17.9).

Листинг 17.8. Синхронизация потоков

```
class MonitorTest
{
  public void TestFunction()
  {
    try
    {
      Monitor.Enter(this);
      // далее – код ресурса
    }
    finally
    {
      Monitor.Exit(this);
    }
  }
}
```

Листинг 17.9. Ключевое слово `lock`

```
class MonitorTest
{
  public void TestFunction()
  {
    lock(this)
    {
      // далее - код ресурса
    }
  }
}
```

17.8. Синхронизация потоков: "один пишет, многие читают"

Часто необходим доступ к ресурсу, запись в который происходит из одного потока, но редко, а чтение — из многих потоков. В таких случаях существует лучшее решение, чем использование класса `Monitor`. Класс `System.Threading.ReaderWriterLock` позволяет создавать кратковременные блокировки для чтения и записи.

17.9. Использование *AutoResetEvent*

При необходимости "будить" поток в определенные моменты времени можно использовать класс синхронизации `System.Threading.AutoResetEvent` (листинг 17.11). Класс потока приведен в листинге 17.10.

Методы класса `WaitHanle` позволяют ожидать не только одного события, но и нескольких одновременно, а также некоторого тайм-аута. Например:

```
// Спать, пока не будет сигнала или не пройдет 5 секунд
WaitHandle.WaitAny(new AutoResetEvent[]{ev}, 5000, false);
```

Аналогично методу `WaitAny`, метод `WaitAll` позволяет дождаться наступления всех необходимых событий или наступления тайм-аута.

Листинг 17.10. Класс потока для листинга 17.11

```
using System;
using System.Threading;

namespace Threads_Events
{
  public class ThreadClass
```

```
  {
    private bool Stop;
    private AutoResetEvent ev;

    public ThreadClass()
    {
      Stop = false;
      ev = new AutoResetEvent(false);
    }

    public void Signal()
    {
      ev.Set();
    }

    // Основной метод потока
    public void Execute()
    {
      // Основной цикл потока
      while (!Stop)
      {
        try
        {
          for (int i=0; i<1000; i++)
          {
            Console.WriteLine(string.Format("Выполнение
                                          потока i={0}", i));
          }

          // Спать, пока не будет сигнала
          ev.WaitOne();
        }
        catch (ThreadInterruptedException)
        {
          Stop = true;
        }
      }

      Console.WriteLine("Выполнение потока завершено.");
    }
  }
}
```

Листинг 17.11. Использование `AutoResetEvent`

```
using System;
using System.Threading;
using System.Diagnostics;

namespace Threads_Events
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Создаем класс потока
      ThreadClass tc = new ThreadClass();
      // Создаем сам поток
      Thread t = new Thread(new ThreadStart(tc.Execute));
      // Старт потока
      t.Start();

      // Поток ждет нашего сигнала
     Console.ReadLine();
     tc.Signal();

     // Завершаем поток
      Console.ReadLine();
      t.Interrupt();

      // Просто выходим по нажатию клавиши <Enter>
      Console.WriteLine("Нажмите Enter для выхода...");
      Console.ReadLine();
    }
  }
}
```

ГЛАВА 18

Процессы

18.1. Запуск внешних программ

Запуск внешней программы производится с помощью класса `System.Diagnostics.Process` (листинг 18.1). При необходимости дождаться завершения работы приложения можно использовать событие `Exited`.

Листинг 18.1. Запуск внешней программы

```
using System;

namespace CreateNewProcess
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Создать новый процесс
      System.Diagnostics.Process proc =
                          new System.Diagnostics.Process();
      // Приложение, которое будем запускать
      proc.StartInfo.FileName = "Notepad.exe";
      proc.EnableRaisingEvents = true;
      proc.Exited +=new EventHandler(proc_Exited);
      proc.Start();

      Console.ReadLine();
    }
```

```
    // Будет вызываться при завершении программы Блокнот
    private static void proc_Exited(object sender, EventArgs e)
    {
      Console.WriteLine("proc_Exited");
    }
  }
}
```

18.2. Блокировка запуска второй копии программы

18.2.1. Вариант 1

Один из способов — создание именованного объекта мьютекса (mutex). Если при старте программы мьютекс с таким именем уже существует, значит, программа уже загружена (листинг 18.2).

Листинг 18.2. Блокировка второй копии программы (вариант 1)

```
using System;
using System.Threading;

namespace OneInstanceApplication1
{
  class Class1
  {
    static Mutex m_mutex;

    [STAThread]
    static void Main(string[] args)
    {
      if (InstanceExists())
      {
        return;
      }
      Console.WriteLine("Приложение запущено.
                         Нажмите ENTER для выхода");
      Console.ReadLine();
    }

    static bool InstanceExists()
    {
      bool createdNew;
```

```
      m_mutex = new Mutex(false, "OneInstanceApplication",
                          out createdNew);
      return(!createdNew);
    }
  }
}
```

18.2.2. Вариант 2

Еще один способ проверки существования копии программы — проверка всех процессов (листинг 18.3).

Листинг 18.3. Блокировка второй копии программы (вариант 2)

```
using System;
using System.Diagnostics;
using System.Reflection;

namespace OneInstanceApplication2
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      Process process = RunningInstance();
      if (process != null)
      {
        Console.WriteLine("Программа уже запущена");
        return;
      }

      Console.WriteLine("Приложение запущено.
                         Нажмите ENTER для выхода");
      Console.ReadLine();
    }

    public static Process RunningInstance()
    {
      Process current = Process.GetCurrentProcess();
      Process[] processes =
                 Process.GetProcessesByName (current.ProcessName);
```

```
      // Просматриваем все процессы
      foreach (Process process in processes)
      {
        // Игнорируем текущий процесс
        if (process.Id != current.Id)
        {
          // Проверяем, что процесс запущен из того же файла
          if (Assembly.GetExecutingAssembly().Location.Replace(
                        "/", "\\") == current.MainModule.FileName)
          {
            // Да, это и есть копия нашего приложения
            return process;
          }
        }
      }
      // Нет, таких же процессов не найдено
      return null;
    }
  }
}
```

18.3. Информация о процессах

Перечисление всех процессов:

```
foreach ( System.Diagnostics.Process p
          in System.Diagnostics.Process.GetProcesses() )
  Console.WriteLine( p.ToString() );
```

Класс `Process` позволяет получить множество информации о процессе, например: `Handle`, `HandleCount`, `ProcessName`, `StartTime`, `UserProcessorTime` и т. д. Также можно вызвать метод `Kill()` для остановки указанного процесса.

Метод `System.Diagnostics.Process.GetProcesses()` может принимать параметр — имя компьютера, на котором производится перечисление процессов. Разумеется, эта операция требует необходимых прав пользователя.

Листинг 18.4 показывает отображение информации о процессах, как в диспетчере задач Windows.

Листинг 18.4. Отображение информации о процессах (как в диспетчере задач)

```
using System;
using System.Diagnostics;
```

```
namespace ProcessesInformation
{
  class SampleProcessMgmt
  {
    static void Main()
    {
      // Получаем список процессов на локальной машине
      Process[] procs = Process.GetProcesses();

      // Можно получить список процессов на удаленной машине
      // Process[] procs = Process.GetProcesses(имя машины);

      // "Рисуем" столбцы как в диспетчере задач
      Console.Write(Print("Image Name"));
      Console.Write(Print("PID"));
      Console.Write(Print("CPU Time"));
      Console.Write(Print("MEM Usage"));
      Console.Write(Print("Peak MEM Usage"));
      Console.Write(Print("Handles"));
      Console.Write(Print("Threads"));
      Console.WriteLine();

      // Проходим по всем процессам
      foreach(Process proc in procs)
      {
        // Обновляем информацию о процессе
        proc.Refresh();

        string name = proc.ProcessName;
        string pid = proc.Id.ToString();

        // Процессорное время в формате hhh:mm:ss
        TimeSpan cputime = proc.TotalProcessorTime;
        string time = String.Format(
          "{0}:{1}:{2}",
          ((cputime.TotalHours-1<0?
                   0:cputime.TotalHours-1)).ToString("##0"),
          cputime.Minutes.ToString("00"),
          cputime.Seconds.ToString("00")
          );

        string mem  = (proc.WorkingSet/1024).ToString()+"k";
        string peakmem = (proc.PeakWorkingSet/1024).ToString()+"k";
```

```
        string handles = proc.HandleCount.ToString();
        string threads = proc.Threads.Count.ToString();

        // Отображаем
        Console.Write(Print(name));
        Console.Write(Print(pid));
        Console.Write(Print(time));
        Console.Write(Print(mem));
        Console.Write(Print(peakmem));
        Console.Write(Print(handles));
        Console.Write(Print(threads));
        Console.WriteLine();

        proc.Close();
      }
      procs = null;
    }

    // Форматированный вывод
    static string Print(string str)
    {
      int maxlen = 16;
      if (str.Length >= maxlen)
      {
        return string.Format("{0}...",
                             str.Substring(0, maxlen - 4));
      }
      str += new string(' ', maxlen - str.Length);
      return str;
    }
  }
}
```

18.4. Определение всех процессов, использующих .NET

Кирк Алиен (Kirk Alien) дает такой совет: для определения всех процессов, использующих .NET, используйте команду

```
tasklist /m "mscor*"
```

18.5. Перечисление всех оконных процессов

Перечисление всех оконных процессов представлено в листинге 18.5.

Листинг 18.5. Перечисление всех оконных процессов

```
using System;
using System.Diagnostics;

namespace ShowAllGUIProcesses
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      foreach (Process p
           in Process.GetProcesses(System.Environment.MachineName))
      {
        if(p.MainWindowHandle != IntPtr.Zero)
        {
          Console.WriteLine(p.ToString());
        }
      }
    }
  }
}
```

18.6. Запуск внешнего процесса и вывод результата на консоль (Windows NT)

С помощью класса `Process` можно запустить внешний процесс и перехватить его вывод (листинг 18.6).

Листинг 18.6. Запуск внешнего процесса и вывод результата на консоль

```
using System;
using System.Diagnostics;
using System.IO;

namespace RunProcess
{
  class Class1
```

```
{
  [STAThread]
  static void Main(string[] args)
  {
    ProcessStartInfo startinfo;
    Process          process = null;
    OperatingSystem  os;
    string           command, stdoutline;
    StreamReader     stdoutreader;

    // Команда, которая будет выполняться
    command = "NETSTAT -a";

    try
    {
      // Работаем, только если это Windows NT
      os = Environment.OSVersion;
      if (os.Platform != PlatformID.Win32NT)
      {
        throw new PlatformNotSupportedException(
           "Требуется Windows NT или выше");
      }
      os = null;

      // Проверка
      if (command == null || command.Trim().Length == 0)
      {
        throw new ArgumentNullException("command");
      }

      startinfo = new ProcessStartInfo();
      // Запускаем через cmd
      startinfo.FileName = "cmd.exe";
      // Ключ /c - выполнение команды
      startinfo.Arguments = "/C " + command;
      // Не используем shellexecute
      startinfo.UseShellExecute = false;
      // Перенаправить вывод на обычную консоль
      startinfo.RedirectStandardOutput = true;
      // Не надо окон
      startinfo.CreateNoWindow = true;
      // Стартуем
      process = Process.Start(startinfo);
```

```
        // Получаем вывод
        stdoutreader = process.StandardOutput;
        while((stdoutline=stdoutreader.ReadLine())!= null)
        {
          // Выводим
          Console.WriteLine(stdoutline);
        }
        stdoutreader.Close();
        stdoutreader = null;
      }
      catch
      {
        throw;
      }
      finally
      {
        if (process != null)
        {
          // Закрываем
          process.Close();
        }

        // Освобождаем
        process = null;
        startinfo = null;
      }
      Console.ReadLine();
    }
  }
}
```

18.7. Получение списка модулей, связанных с текущим процессом

В листинге 18.7 показано, как получить имена всех модулей, связанных с текущим процессом.

Листинг 18.7. Получение списка модулей, связанных с текущим процессом

```
using System;
using System.Diagnostics;
using System.IO;
```

```
namespace ProcessReference
{
  class Class1
  {
    static void Main()
    {
      Process p;

      // Получаем текущий процесс
      p = Process.GetCurrentProcess();

      // Выводим все DLL, связанные с ним
      foreach(ProcessModule module in p.Modules)
      {
        Print(module.ModuleName, 20);
        Print(module.FileName  , 75);
        Print(module.FileVersionInfo.FileVersion, 20);
        Print(File.GetLastWriteTime(module.FileName).
                                    ToShortDateString(), 20);
        Console.WriteLine();
      }
      p.Close();
      p = null;
      Console.ReadLine();
    }

    // Специальная функция форматирования строки
    static void Print(string towrite, int maxlen)
    {
      if (towrite.Length >= maxlen)
      {
        Console.Write(towrite.Substring(0, maxlen - 4)+"... ");
        return;
      }
      towrite += new string(' ', maxlen - towrite.Length);
      Console.Write(towrite);
    }
  }
}
```

18.8. Управление сервисами

Управление сервисами осуществляется с помощью класса `System.ServiceProcess.ServiceController` (листинг 18.8). Для работы этого кода требуется добавление ссылки на `System.ServiceProcess.dll`.

Листинг 18.8. Управление сервисами

```
using System;
using System.ServiceProcess;

namespace WinServiceManagement
{
  class Class1
  {
    static void Main()
    {
      ServiceController sc;
      string            serviceName;
      TimeSpan          timeout = new TimeSpan(0, 0, 30);
      OperatingSystem   os;

      os = Environment.OSVersion;
      if (os.Platform != PlatformID.Win32NT)
      {
        throw new PlatformNotSupportedException(
              "Для работы нужна ОС Windows NT, 2000, XP или выше");
      }
      os = null;

      // Имя сервиса (например, telnet)
      serviceName = "Telnet";

      // Создаем контроллер
      sc = new ServiceController(serviceName);

      // Проверяем статус процесса
      if (sc.Status == ServiceControllerStatus.Running)
      {
        // Процесс можно остановить?
        if (sc.CanStop)
        {
          Console.WriteLine("Останов сервиса {0}", serviceName);
          sc.Stop();

          try
          {
            // Подождем 30 секунд
            sc.WaitForStatus(
                    ServiceControllerStatus.Stopped, timeout);
```

```
            Console.WriteLine(
                "Сервис {0} успешно остановлен", serviceName);
          }
          catch(TimeoutException)
          {
            Console.WriteLine(
                "Не удалось остановить сервис {0}", serviceName);
          }
        }
        else
        {
          Console.WriteLine(
               "Сервис {0} не может быть остановлен", serviceName);
        }
      }

      // Если сервис остановлен - запустим его
      if (sc.Status == ServiceControllerStatus.Stopped)
      {
        // Стартуем
        sc.Start();

        try
        {
          // Ждем 30 секунд
          sc.WaitForStatus(
                 ServiceControllerStatus.Running, timeout);
          Console.WriteLine(
                 "Сервис {0} успешно стартован", serviceName);
        }
        catch(TimeoutException)
        {
          Console.WriteLine(
                "Не удалось запустить сервис {0}", serviceName);
        }
      }

      // Закрыть контроллер
      sc.Close();
      sc = null;
    }
  }
}
```

ГЛАВА 19

Разное

19.1. Перечисление всех принтеров системы

Доступ к списку установленных принтеров производится с помощью библиотеки `IWshRuntimeLibrary` (листинг 19.1).

Листинг 19.1. Перечисление всех принтеров системы

```
using System;
using System.Collections;
using IWshRuntimeLibrary;

namespace EnumPrinterConnections
{
  class AllPrintersClass
  {
    [STAThread]
    static void Main(string[] args)
    {
      WshNetwork network = new WshNetwork();
      foreach (IEnumerable printer in
                          network.EnumPrinterConnections())
      {
        Console.WriteLine(printer.ToString());
      }
    }
  }
}
```

19.2. Установка принтера по умолчанию

Установка принтера по умолчанию производится с помощью библиотеки `IWshRuntimeLibrary` (листинг 19.2).

Листинг 19.2. Установка принтера по умолчанию

```
using System;
using IWshRuntimeLibrary;

namespace SetDefaultPrinter
{
  class DefaultPrinter
  {
    [STAThread]
    static void Main(string[] args)
    {
      WshNetwork network = new WshNetwork();

       // Получить список принтеров
      IWshCollection Printers = network.EnumPrinterConnections();
      if (Printers.Count() > 0)
      {
        // Индекс устанавливаемого принтера в списке принтеров
        object index = (object)"1";
        // Установка принтера с выбранным индексом
        // как принтера по умолчанию
        network.SetDefaultPrinter((string)Printers.Item(ref index));
      }
    }
  }
}
```

19.3. Определение версии операционной системы

Версию операционной системы можно получить с помощью класса `Environment.OSVersion`:

```
string PlatformText = Environment.OSVersion.Platform.ToString();
string VersionText  = Environment.OSVersion.Version.ToString();
Console.WriteLine(PlatformText + " " + VersionText);
```

19.4. Получение текущей даты и времени

Для получения текущей даты и времени предназначено свойство `DateTime.Now`. Для получения текущей даты используется свойство `DateTime.Today`. В отличие от `DateTime.Now`, это свойство возвращает величину `DateTime` с установленным в ноль временем.

19.5. Установка значения десятичного разделителя

Установить значение десятичного разделителя напрямую нельзя, т. к. нельзя модифицировать `CurrentCulture`. Для установки значения десятичного разделителя надо создать новый экземпляр (например, клонировав существующий) и изменить нужные поля (листинг 19.3).

Листинг 19.3. Установка значения десятичного разделителя

```
using System;
using System.Globalization;
using System.Threading;

namespace DecimalSeparator
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      Console.WriteLine(1.5F); // Выведет 1.5

      // Нельзя напрямую менять CurrentCulture
      // CultureInfo.CurrentCulture.NumberFormat.
      //                 NumberDecimalSeparator= "#";

      // Необходимо создать новый экземпляр CultureInfo
      // и изменить в нем нужные поля
      CultureInfo newCInfo =
          (CultureInfo) Thread.CurrentThread.CurrentCulture.Clone();
      newCInfo.NumberFormat.NumberDecimalSeparator = "#";
      Thread.CurrentThread.CurrentCulture = newCInfo;
      Console.WriteLine(1.5F); // Выведет 1#5
    }
  }
}
```

19.6. Определение числа процессоров

Код, представленный в листинге 19.4, определяет число процессоров.

Листинг 19.4. Определение числа процессоров

```
using System;
using System.Diagnostics;
namespace ProcessorCount
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      int am =
           Process.GetCurrentProcess().ProcessorAffinity.ToInt32();
      int processorCount = 0;
      while(am != 0)
      {
        processorCount++;
        am &= (am - 1);
      }
      Console.WriteLine(processorCount);
    }
  }
}
```

19.7. Запись в системный журнал событий

Запись в системный журнал событий (application log) производится с помощью методов класса `System.Diagnostics.EventLog`:

```
EventLog.WriteEntry(Имя_источника, строка_сообщения, тип);
```

Например:

```
EventLog.WriteEntry("Threads", Name, EventLogEntryType.Warning);
```

19.8. Создание своего журнала событий

Журнал событий (event log) — удобное средство для записи информации о ходе работы программы или сервиса. Листинг 19.5 показывает работу с несистемным журналом событий.

Листинг 19.5. Создание своего журнала событий

```
using System;
using System.Diagnostics;

namespace EventLogTest
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      EventLog elog;

      // Имя программы, связанной с журналом
      string eventSource = "EventLogTest";

      // Имя журнала (должно быть уникальным и не совпадать
      // с системными именами, такими как Application,
      // Security или System)
      string logName = "test app debug log";

      // Если программа еще не зарегистрирована,
      // то регистрируем
      if (!EventLog.SourceExists(eventSource))
      {
        EventLog.CreateEventSource(eventSource, logName);
      }

      // Объект для работы с журналом
      elog = new EventLog();
      elog.Log = logName;
      elog.Source = eventSource;
      elog.EntryWritten +=
                    new EntryWrittenEventHandler(OnEntryWritten);
      elog.EnableRaisingEvents=true;

      // Ограничиваем число записей
      if (elog.Entries.Count > 20)
      {
        // Чистим журнал
        elog.Clear();
        Console.WriteLine("Лог {0} очищен", logName);
      }
```

```
      // Записываем строку с типом Information
      elog.WriteEntry("event informational text",
                              EventLogEntryType.Information);
      // Записываем строку с типом Error
      elog.WriteEntry("event error text", EventLogEntryType.Error);
      // Записываем строку с типом Warning
      elog.WriteEntry("event warning text",
                              EventLogEntryType.Warning);

      // Запись может занять некоторое время - подождем
      System.Threading.Thread.Sleep(2000);

      // Закрываем журнал
      elog.Close();
      elog = null;
    }

    // Обработка события записи в журнал
    protected static void OnEntryWritten(object sender,
                            EntryWrittenEventArgs e)
    {
      Console.WriteLine("Запись в журнал {0}. Время {1}. Позиция {2}.",
        ((EventLog)sender).LogDisplayName,
        e.Entry.TimeWritten, e.Entry.Index);
    }
  }
}
```

19.9. Как "отвязать" программу от журнала событий

При попытке создать журнал событий для программы, которая уже регистрировала один журнал, возникает ошибка — необходимо сначала уничтожить предыдущий журнал (листинг 19.6).

Листинг 19.6. Удаление журнала событий

```
using System;
using System.Diagnostics;

namespace EventLogDelete
{
  class Class1
```

```
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Имя программы, связанной с журналом
      string eventSource = "EventLogTest";

      // Имя журнала (должно быть уникальным и не совпадать
      // с системными именами, такими как Application,
      // Security или System)
      string logName = "test app debug log";

      // Удаляем регистрацию программы
      if (EventLog.SourceExists(eventSource))
      {
        EventLog.DeleteEventSource(eventSource);
      }

      // Удаляем журнал
      if (EventLog.Exists(logName))
      {
        EventLog.Delete(logName);
      }
    }
  }
}
```

19.10. Перечисление доступных журналов событий

Метод `GetEventLogs` позволяет получить список доступных журналов (листинг 19.7).

Листинг 19.7. Перечисление журналов событий

```
using System;
using System.Diagnostics;

namespace EventLogList
{
  class Class1
  {
    [STAThread]
```

```
    static void Main(string[] args)
    {
      EventLog[] elogs = EventLog.GetEventLogs();
      foreach(EventLog elog in elogs)
      {
        Console.WriteLine("{0}", elog.LogDisplayName);
        elog.Close();
      }
      elogs = null;
    }
  }
}
```

19.11. Звуковой сигнал

В .NET 1.1 для подачи звукового сигнала (beep) предназначена функция `Beep` библиотеки `Kernel32` (листинг 19.8).

Для консольных программ можно использовать вывод символа alarm, подающего короткий звуковой сигнал через системный динамик:

```
Console.Write("\a");
```

Листинг 19.8. Звуковой сигнал

```
// Описание
[DllImport("kernel32.dll")]
static extern bool Beep(int freq, int duration);
// Вызов
Beep(1000, 100);
```

Или:

```
// Описание
[DllImport("user32.dll")]
static extern void MessageBeep(int uType);
// Вызов
MessageBeep(0);
```

19.12. Генерация случайного числа

Генерацию случайного числа можно выполнить так:

```
// Создание генератора случайных чисел со случайной инициализацией
Random r = new Random( DateTime.Now.Millisecond );
```

```
// Генерация случайного числа
int value1 = r.Next();
// Генерация случайного числа в указанном диапазоне
int value2 = r.Next(1, 10);
```

19.13. Принудительная сборка мусора

Принудительный запуск сборщика мусора выполняется с помощью следующих строк кода:

```
// Запуск сборщика мусора
GC.Collect();
// Ожидание завершения работы всех Finiliaze-методов
GC.WaitForPendingFinalizers();
// Запуск сборщика мусора
GC.Collect();
```

Это достаточно дорогостоящая операция, поэтому ее следует выполнять только в случаях крайней необходимости.

Если же даже такая операция не помогает удалить объект из памяти, можно выполнить операцию

```
GC.GetTotalMemory(true);
```

Эта строка запускает сборщик мусора до 20 раз, пытаясь уменьшить разницу свободной памяти между проходами до 5%.

19.14. Получение пути к папке Framework

Получение пути к папке Framework можно осуществить с помощью вызова функции `GetCORSystemDirectory` (листинг 19.9).

Листинг 19.9. Получение пути к папке Framework

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

namespace GetFrameworkPathApplication
{
  class MainClass
  {
    [DllImport("mscoree.dll")]
    internal static extern void GetCORSystemDirectory (
       [MarshalAs(UnmanagedType.LPTStr)]
       System.Text.StringBuilder Buffer,
```

```
       int BufferLength, ref int Length
    );

    [STAThread]
    static void Main(string[] args)
    {
      System.Text.StringBuilder sb =
                  new System.Text.StringBuilder(1024);
      int size = 0;

      // Вызов
      GetCORSystemDirectory(sb, sb.Capacity, ref size);

      // Напечатает, например,
      // "F:\WINXP\Microsoft.NET\Framework\v1.1.4322\"
      Console.WriteLine(sb);
    }
  }
}
```

19.15. Проигрывание WAV-файла

Для проигрывания звуковых файлов можно использовать метод `PlaySound` из звуковой библиотеки Windows, как показано в листинге 19.10.

Листинг 19.10. Проигрывание WAV-файла

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

namespace PlayWav
{
  class Class1
  {
    [DllImportAttribute("winmm.dll")]
    public static extern long PlaySound (String lpszName,
                          long hModule, long dwFlags);

    [STAThread]
    static void Main(string[] args)
    {
      PlaySound(@"C:\WINDOWS\Media\recycle.wav", 0, 0);
    }
  }
}
```

19.16. Чтение реестра

Для обращения к реестру используются классы библиотеки `Microsoft.Win32` (листинг 19.11).

Листинг 19.11. Чтение реестра

```
using System;
using Microsoft.Win32;
namespace ReadRegistry
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      RegistryKey hklm = Registry.LocalMachine;
      hklm=hklm.OpenSubKey("HARDWARE\\DESCRIPTION\\System\\
                                    CentralProcessor\\0");
      Object obp=hklm.GetValue("Identifier");
      Console.WriteLine("Processor Identifier :{0}",obp);
      RegistryKey hklp =Registry.LocalMachine;
      hklp=hklp.OpenSubKey("HARDWARE\\DESCRIPTION\\System\\
                            CentralProcessor\\0");
      Object obc=hklp.GetValue("VendorIdentifier");
      Console.WriteLine("Vendor Identifier    :{0}",obc);
      RegistryKey biosv =Registry.LocalMachine;
      biosv=biosv.OpenSubKey("HARDWARE\\DESCRIPTION\\System\\
                            MultiFunctionAdapter\\4");
      Object obv=biosv.GetValue("Identifier");
      Console.WriteLine("Bios Status          :{0}",obv);
      RegistryKey biosd =Registry.LocalMachine;
      biosd=biosd.OpenSubKey("HARDWARE\\DESCRIPTION\\System\\");
      Object obd=biosd.GetValue("SystemBiosDate");
      Console.WriteLine("Bios Date            :{0}",obd);
      RegistryKey bios =Registry.LocalMachine;
      bios=bios.OpenSubKey("HARDWARE\\DESCRIPTION\\System\\");
      Object obs=bios.GetValue("Identifier");
      Console.WriteLine("System Identifer     :{0}",obs);
    }
  }
}
```

19.17. Разрешение обращения к реестру

Для разрешения обращения к реестру в файл AssemblyInfo.cs должны быть добавлены соответствующие атрибуты (листинг 19.12).

Листинг 19.12. Атрибуты для разрешения обращения к реестру

```
using System.Reflection;
using System.Runtime.CompilerServices;
using Microsoft.Win32;
using System.Security.Permissions;
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
// Обеспечение прав на обращение к ключу HKEY_CURRENT_USER
[assembly:
RegistryPermissionAttribute(SecurityAction.RequestMinimum,
                            All = "HKEY_CURRENT_USER")]
```

19.18. Сохранение объекта в реестре

Для сохранения объекта в реестре объект должен быть сериализуемым, т. е. либо иметь атрибут `[Serializable]`, либо быть наследником интерфейса `ISerializable` (листинг 19.13).

Для разрешения обращения к реестру в файле AssemblyInfo.cs должны быть добавлены соответствующие строки:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using Microsoft.Win32;
using System.Security.Permissions;
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
// Обеспечение прав на обращение к ключу HKEY_CURRENT_USER
[assembly: RegistryPermissionAttribute(SecurityAction.RequestMinimum,
  All = "HKEY_CURRENT_USER")]
```

Листинг 19.13. Сохранение объекта в реестре

```
using System;
using System.Collections;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using Microsoft.Win32;
using System.Security.Permissions;
```

```
namespace SerializeTest
{
  class TestClass
  {

    static void Save(object obj, string akey, string avalue)
    {
      // Создание бинарного потока
      BinaryFormatter formatter = new BinaryFormatter();
      MemoryStream stream = new MemoryStream();

      // Сериализация в бинарный поток
      formatter.Serialize(stream, obj);

      // Описание ключа реестра
      RegistryKey regKey;
      // Открытие ключа реестра
      regKey = Registry.CurrentUser.CreateSubKey(akey);
      // Запись в реестр
      regKey.SetValue(avalue, stream.ToArray());
    }

    static void Load(ref object obj, string akey, string avalue)
    {
      // Создание бинарного потока
      BinaryFormatter formatter = new BinaryFormatter();
      MemoryStream stream = new MemoryStream();

      // Описание ключа реестра
      RegistryKey regKey;
      // Открытие ключа реестра
      regKey = Registry.CurrentUser.OpenSubKey(akey);

      // Чтение из реестра в байтовый массив
      byte[] barray = null;
      barray = (byte[])regKey.GetValue(avalue);

      if(barray != null)
      {
        stream.Write(barray, 0, barray.Length);
        stream.Position = 0;
        obj = formatter.Deserialize(stream) as ArrayList;
      }

    }
```

```
    [STAThread]
    static void Main(string[] args)
    {
      // объект, поддерживающий сериализацию, например, ArrayList
      ArrayList names = new ArrayList();

      // Инициализация объекта
      names.Add("1");
      names.Add("2");
      names.Add("3");

      // Сохранение
      Save(names, "test", "ValueName");

      // Загружаем сохраненный объект
      object obj = null;
      Load(ref obj,"test", "ValueName");
      foreach (string str in (obj as ArrayList))
      {
        Console.WriteLine(str);
      }
    }
  }
}
```

19.19. Определение типа файла по расширению

Тип файла можно получить из реестра (листинг 19.14). Примеры типов файлов *(см. также разд. 24.5)*:

- `text/plain` — обычный текст;
- `image/bmp` — изображение формата BMP;
- `application/pdf` — PDF-файл;
- `image/gif` — изображение формата GIF;
- `image/jpeg` — изображение формата JPEG;
- `text/xml` — документ в формате XML;
- `application/msexcel` — файл MS Excel;
- `application/msword` — файл MS Word.

Листинг 19.14. Определение типа файла по расширению

```
using System;
using Microsoft.Win32;

namespace FileTypeByExt
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Тип (content) по умолчанию
      const string DEFAULT_CONTENT_TYPE = "application/unknown";

      string  fileContentType;

      // Расширение файла
      string fileExtension = ".jpeg";

      try
      {
        // Ищем в реестре ветку, соответствующую расширению
        RegistryKey fileextkey =
                   Registry.ClassesRoot.OpenSubKey(fileExtension);
        // Получаем тип
        fileContentType = fileextkey.GetValue("Content Type",
                        DEFAULT_CONTENT_TYPE).ToString();
      }
      catch (Exception e)
      {
        fileContentType = DEFAULT_CONTENT_TYPE;
        Console.WriteLine(e.Message);
      }

      Console.WriteLine(fileContentType);
    }
  }
}
```

19.20. Создание GUID

Класс `System.Guid` позволяет сгенерировать строку GUID (листинг 19.15).

Листинг 19.15. Создание GUID

```
using System;
namespace GenerateGUID
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      System.Guid guid = System.Guid.NewGuid();
      Console.WriteLine(guid.ToString());
    }
  }
}
```

19.21. Блокировка компьютера

Код, приведенный в листинге 19.16, блокирует компьютер так же, как будто была нажата комбинация клавиш <⊞>+<L>.

Листинг 19.16. Блокировка компьютера (как по комбинации клавиш <⊞>+<L>)

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

namespace LockUp
{
  class Class1
  {
    [DllImport("user32.dll")]
    public static extern void LockWorkStation ();

    [STAThread]
    static void Main(string[] args)
    {
      LockWorkStation();
    }
  }
}
```

19.22. Как создать COM-объект по GUID?

Создание COM-объекта по GUID:

```
Activator.CreateInstance(Type.GetTypeFromCLSID(clsid));
```

19.23. Почему уничтожается remoting-объект?

По умолчанию, если к методам remoting-объекта 5 минут не обращаются, он считается ненужным и выгружается. Чтобы изменить такое поведение, можно сделать, например, так (внутри тела класса remoting-объекта):

```
public override object InitializeLifetimeService()
{
  return null;
}
```

19.24. Ошибка *SerializationException*: "BinaryFormatter Version incompatibility" при использовании .NET Remoting

При использовании .NET Remoting иногда возникает ошибка

```
[SerializationException: BinaryFormatter Version incompatibility.
Expected Version 1.0 Received Version 1835627630.1699884645.]
```

Эта ошибка связана с тем, что клиент получает не бинарные данные, а простой HTML-текст, из которого `BinaryFormatter` пытается получить заголовок пакета. Соответственно вместо реального номера версии получается некое странное число. Посылка HTML-текста вместо бинарного потока чаще всего связана с ошибкой, которую IIS посылает в виде HTML, например, ошибка 404 (файл не найден) или 500 (внутренняя ошибка сервера):

```
HTTP/1.1 500 Internal Server Error
Server: Microsoft-IIS/5.0
Date: Mon, 22 Apr 2002 21:25:44 GMT
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 3286
```

Более редкий случай — ошибка в настройках .NET Remoting, когда клиент ожидает данные в бинарном формате, а сервер отправляет их с помощью протокола SOAP.

Свой вариант реализации remoting-канала предлагает Ричард Блеветт (Richard Blewett): **www.dotnetconsult.co.uk/weblog/permalink.aspx/827189d3-ee0e-444f-b01d-bf9ce9f70f5c**.

19.25. Использование Speech API

Для воспроизведения текста голосом можно воспользоваться библиотекой Microsoft Speech (листинг 19.17). Для этого к проекту нужно подключить COM-объект Microsoft Speech Object.

Листинг 19.17. Использование Speech API

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Threading;
using System.Windows.Forms;
using System.Data;
using SpeechLib;

namespace SAPI5
{
  public class Form1 : System.Windows.Forms.Form
  {
    private System.Windows.Forms.TextBox textBox1;
    private System.Windows.Forms.Panel panel1;
    private System.Windows.Forms.Button btnExit;
    private System.Windows.Forms.Button btnSpeak;
    private System.Windows.Forms.CheckBox cbSaveToWave;
    private System.Windows.Forms.ComboBox cbVoices;
    private System.Windows.Forms.TrackBar Volume;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.TrackBar Rate;

    private System.ComponentModel.Container components = null;

    // Основной объект SAPI
    SpVoice m_voice;

    public Form1()
    {
      InitializeComponent();
    }

    protected override void Dispose( bool disposing )
    {
      if( disposing )
```

```
    {
      if (components != null)
      {
        components.Dispose();
      }
    }
    base.Dispose( disposing );
  }

  #region Windows Form Designer generated code
  ... ... ... ... ... ... ... ... ... ... ...
  #endregion

  [STAThread]
  static void Main()
  {
    Application.Run(new Form1());
  }

  private void btnSpeak_Click(object sender, System.EventArgs e)
  {
    if ((Convert.ToInt32(btnSpeak.Tag) == 0) && (sender != null))
    {
      // Говорить
      btnSpeak.Tag = 1;
      btnSpeak.Text = "STOP";

      try
      {
        // Если выбрана звуковая схема
        if (cbVoices.Text.Length >0)
        {
          // Установить звуковую схему
          m_voice.Voice = m_voice.GetVoices(
                  string.Format("Name={0}", cbVoices.Text),
                  "Language=409").Item(0);
        }

        // Записывать в файл
        if (cbSaveToWave.Checked)
        {
          SaveFileDialog sfd = new SaveFileDialog();
          sfd.Filter =
             "All files (*.*)|*.*|wav files (*.wav)|*.wav";
```

```
        sfd.Title = "Save to a wave file";
        sfd.FilterIndex = 2;
        sfd.RestoreDirectory = true;
        if (sfd.ShowDialog()== DialogResult.OK)
        {
          SpeechStreamFileMode SpFileMode =
               SpeechStreamFileMode.SSFMCreateForWrite;
          SpFileStream SpFileStream = new SpFileStream();
          SpFileStream.Open(sfd.FileName, SpFileMode, false);
          m_voice.AudioOutputStream = SpFileStream;
          m_voice.Speak(textBox1.Text,
               SpeechVoiceSpeakFlags.SVSFlagsAsync);
          m_voice.WaitUntilDone(Timeout.Infinite);
          SpFileStream.Close();
        }
      }
      else
      {
        // Просто говорить
        m_voice.Speak(
           textBox1.Text, SpeechVoiceSpeakFlags.SVSFlagsAsync);
      }
    }
    catch
    {
      MessageBox.Show("Ошибка");
    }
  }
  else
  {
    // Остановить воспроизведение
    btnSpeak.Tag = 0;
    btnSpeak.Text = "SPEAK";
    m_voice.Speak(null, (SpeechVoiceSpeakFlags)2);
    m_voice.Resume();
  }

}

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

```
    private void Form1_Load(object sender, System.EventArgs e)
    {
      m_voice = new SpVoice();
      m_voice.EndStream += new
           _ISpeechVoiceEvents_EndStreamEventHandler(
                                          m_voice_EndStream);
      // Получить все звуковые схемы
      foreach(ISpeechObjectToken t in m_voice.GetVoices("",""))
      {
        cbVoices.Items.Add(t.GetAttribute("Name"));
      }
    }

    // Задание громкости
    private void Volume_Scroll(object sender, System.EventArgs e)
    {
      m_voice.Volume = Volume.Value;
    }

    // Задание скорости воспроизведения
    private void Rate_Scroll(object sender, System.EventArgs e)
    {
      m_voice.Rate = Rate.Value;
    }

    // Событие "конец воспроизведения"
    private void m_voice_EndStream(
                      int StreamNumber, object StreamPosition)
    {
      btnSpeak_Click(null, null);
    }
  }
}
```

ГЛАВА 20

Регулярные выражения

20.1. Поиск совпадения

Метод `System.Text.RegularExpressions.Regex.IsMatch` возвращает `true`, если строка совпадает с указанным регулярным выражением:

```
string text  =...; // текст
string regstr=...; // регулярное выражение
if (System.Text.RegularExpressions.Regex.IsMatch(text, regstr))
  Console.WriteLine("Совпадает");
else
  Console.WriteLine("Не совпадает");
```

Некоторые регулярные выражения приведены в табл. 20.1. Некоторые строки начинаются с символа `@` для того, чтобы не усложнять конструкцию регулярных выражений дополнительными слэшами.

Таблица 20.1. *Примеры регулярных выражений*

Регулярное выражение	Значение	Комментарии
`@"^\s*$"`	Пустая строка	• `^` — начало строки; • `\s` — разделитель; • `$` — конец строки
`"^[0-9]*$"`	Только цифры	• `^` — начало строки; • `[0-9]` — любой символ из диапазона 0—9; • `$` — конец строки

Таблица 20.1 *(окончание)*

Регулярное выражение	Значение	Комментарии
`"[a-ZA-Z_][a-zA-Z_0-9]*"`	Имя переменной	Имя начинается с буквы или подчеркивания, затем следуют буквы, цифры или знаки подчеркивания (символ `*` обозначает "ни одного, один или более предыдущих выражений")
`"[a-ZA-Z_][a-zA-Z_0-9]{0,31}"`	Имя переменной, не более 32 символов	Аналогично предыдущему выражению, но символ `*` заменен выражением `{0,31}`, обозначающим число возможных повторений
`"(1[012]|[1-9]):[0-5][0-9]?(am|pm)"`	Время в формате AM/PM	Символ `|` означает выбор одного из вариантов. Символ `?` означает повторение "или один раз или ни одного"
`"0?[0-9]|1[0-9]|2[0-3]"` или короче `"[01]?[0-9]|2[0-3]"`	Время в 24-часовом формате	Возможны три интервала: 00—09, 10—19 и 20—23

20.2. Поиск и получение совпавшего текста

Метод `Match` возвращает текст, совпавший с регулярным выражением. Если совпадение отсутствует, возвращается пустая строка.

Пример — выделение числа из строки:

```
string text = "Температура равна 19 градусам";
System.Text.RegularExpressions.Match match =
     System.Text.RegularExpressions.Regex.Match(text, @"\d+");
Console.WriteLine(match.ToString());   // напечатает 19
```

При необходимости найти все совпадения используется метод `Matches`:

```
string text = "Температура равна 19 градусам. Завтра будет 20.";
System.Text.RegularExpressions.MatchCollection matchs =
   System.Text.RegularExpressions.Regex.Matches(text, @"\d+");
foreach (System.Text.RegularExpressions.Match m in matchs)
{
  Console.WriteLine(m.ToString()); // напечатает 19 и 20
}
```

Все совпадения можно найти и с помощью метода `Match`, вызывая метод `NextMatch()` до тех пор, пока свойство `Success` будет возвращать `true` *(см. разд. 20.4)*.

20.3. Поиск с заменой

Поиск с заменой осуществляется с помощью метода `Replace` класса `System.Text.RegularExpressions.Regex` (листинг 20.1).

Листинг 20.1. Замена тегов .. на .. с помощью регулярных выражений

```
// Строка
string text = @"Итак, <b>поиск с заменой</b> осуществляется "+
              "с помощью метода <b>Replace</b>";
// Выполнение замены
text = System.Text.RegularExpressions.Regex.Replace(
    // строка
    text,
    // регулярное выражение для поиска
    @"<b>(.*?)</b>",
    // регулярное выражение для замены
    @"<I>$1</I>",
    // Параметры замены — игнорировать регистр символов
   System.Text.RegularExpressions.RegexOptions.IgnoreCase
   );
Console.WriteLine(text);
```

20.4. Разделение строки на слова

Для разделения строки на слова можно использовать регулярное выражение `"w+"` (листинг 20.2).

Листинг 20.2. Разделение строки на слова

```
// Будет хранить содержимое файла
string InputStr;

// Открыть файл для чтения
using (StreamReader reader = new StreamReader("input.txt"))
{
  // Прочитать весь файл
  InputStr = reader.ReadToEnd();
}
```

```
Match match = Regex.Match(InputStr, @"\w+");

// Цикл по всем словам
while (match.Success)
{
  // Получаем очередное слово
  string word = match.ToString();

  // Переход к следующему слову
  match = match.NextMatch();
}
```

20.5. Разбор CSV-файла с учетом кавычек

Часто возникает необходимость разобрать строку на части, записанные через разделитель, но при этом учитывать, что разделитель может присутствовать внутри кавычек (листинг 20.3).

Листинг 20.3. Разбор CSV-файла с учетом кавычек

```
using System;
using System.Text.RegularExpressions;

namespace ParseCSV
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      string separator = ";";
      string input     = "AAA;BBB;\"CCC;DDD\";EE";

      string pattern   =
           string.Format("{0}(?=(?:[^\"]*\"[^\"]*\")*(?![^\"]*\"))",
           separator);

      string [] result = Regex.Split(input, pattern);

      // Выведет 4 строки:
      // AAA
      // BBB
      // "CCC;DDD"
      // EE
```

```
      foreach (string str in result)
      {
        Console.WriteLine(str);
      }
    }
  }
}
```

20.6. Выделение макроимен

Листинг 20.4 показывает замену макроимен, указанных в строке с помощью знаков %, на их значения.

Листинг 20.4. Выделение макроимен в файле

```
using System;
using System.Text.RegularExpressions;

namespace MacroNames
{
  class Class1
  {
    static string paramPattern = @"%[^\s%]+%";

    [STAThread]
    static void Main(string[] args)
    {
      // Строка с макропараметрами
      string input = "start %M1% test %M2% end";

      // Разбор строки
      MatchCollection matches = Regex.Matches(input, paramPattern);
      foreach(Match match in matches)
      {
        string paramName = match.Value.Trim('%');
        Console.WriteLine("Имя параметра: {0}", paramName);

        // Здесь как-то вычисляем значение параметра
        string paramValue = string.Format("!{0}!", paramName);

        // Заменяем
        input = Regex.Replace(input, match.Value, paramValue);
      }
```

```
      Console.WriteLine(input);
    }
  }
}
```

20.7. Изменение формата даты

Следующее выражение изменяет формат даты с `mm/dd/yy` на `dd-mm-yy`:

```
string input = "12/24/04";
string output = System.Text.RegularExpressions.Regex.Replace(input,
      "\\b(?<month>\\d{1,2})/(?<day>\\d{1,2})/(?<year>\\d{2,4})\\b",
      "${day}-${month}-${year}");
Console.WriteLine(output); // напечатает 24-12-04
```

20.8. Замена заголовка HTML-файла

Код из листинга 20.5 заменяет заголовок HTML-файла и выводит его на консоль.

Листинг 20.5. Замена заголовка HTML-файла

```
FileStream fs = new FileStream( "test.htm",
                  FileMode.Open, FileAccess.Read );
StreamReader sr = new StreamReader( fs );

Regex r = new Regex( "<TITLE>(.*)</TITLE>" );
string s;
while( (s = sr.ReadLine()) != null )
{
  if( r.IsMatch( s ) )
    s = r.Replace( s, "<TITLE>Новый заголовок файла ${1}</TITLE>" );
  Console.WriteLine( s );
}
```

Описание `${1}` возвращает группу, отмеченную первыми скобками в регулярном выражении, т. е. весь текст между тегами `<TITLE>` и `</TITLE>`.

20.9. Тестирование регулярных выражений

Проверить правильность регулярного выражения можно с помощью многих инструментов, например, Expresso от CodeProject или RegexDesigner от SellsBrothers.com.

20.10. Примеры регулярных выражений

Множество примеров можно найти на сайте **http://regxlib.com**. Некоторые примеры приведены далее.

20.10.1. E-mail

Выражение:

```
(\w[-._\w]*\w@\w[-._\w]*\w\.\w{2,3})
```

Правильно:

```
[foo@bar.com], [foobar@foobar.com.au]
```

Неправильно:

```
[foo@bar], [$$$@bar.com]
```

20.10.2. Положительные десятичные числа

Выражение:

```
(^\d*\.?\d*[1-9]+\d*$)|(^[1-9]+\d*\.\d*$)
```

Правильно:

```
[0.050], [5.0000], [5000]
```

Неправильно:

```
[0], [0.0], [.0]
```

20.10.3. HTML-теги

Выражение:

```
<[^>]+>
```

Правильно:

```
[<html>]
```

Неправильно:

```
[text], [test>]
```

20.10.4. HTML-теги с Java-привязкой

Выражение:

```
<[a-zA-Z][^>]*\son\w+=(\w+|'[^']*'|"[^"]*")[^>]*>
```

Правильно:

```
[<IMG onmouseover="window.close()">]
```

Неправильно:

```
[<IMG src="star.gif">]
```

20.10.5. HTML-теги с атрибутами

Выражение:

```
<([^\s>]*)(\s[^<]*)>
```

Правильно:

```
[<img src="truc">], [<body background='...'>], [<p align='left'>]
```

Неправильно:

```
[<br>], [</body>], [<p>]
```

20.10.6. Строки HTML-цветов

Выражение:

```
^(#){1}([a-fA-F0-9]){6}$
```

Правильно:

```
[#FFFFFF], [#FF3421], [#00FF00]
```

Неправильно:

```
[232323], [f#fddee], [#fd2]
```

20.10.7. XML-теги (закрытые теги)

Выражение:

```
<(\w+)(\s(\w*=".*?")?)*((/>)|((/*?)>.*?</\1>))
```

Правильно:

```
[<body> text<br/>More Text </body>], [<a href="link.html">Link</a]
```

Неправильно:

```
[<p> Some Text <p>], [<hr>], [<html>]
```

20.10.8. MAC-адрес

Выражение:

```
^([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-F])$
```

Правильно:

```
[01:23:45:67:89:ab], [01:23:45:67:89:AB], [fE:dC:bA:98:76:54]
```

Неправильно:

```
[01:23:45:67:89:ab:cd], [01:23:45:67:89:Az], [01:23:45:56:]
```

20.10.9. Имя макроса (формат *@@имя@@*)

Выражение:

```
@{2}((\S)+)@{2}
```

Правильно:

`[@@test@@]`, `[@@name@@]`, `[@@2342@@]`

Неправильно:

`[@test@]`, `[@@na me@@]`, `[@@ name@@]`

20.10.10. Время

Выражение:

```
([0-1][0-9]|2[0-3]):[0-5][0-9]
```

Правильно:

`[00:00]`, `[13:59]`, `[23:59]`

Неправильно:

`[24:00]`, `[23:60]`

ГЛАВА 21

XML

В части примеров используется следующий XML-файл (books.xml):

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<ListOfBooks>
 <Book>
  <Title FontSize="8">Title-1</Title>
  <Price>50</Price>
 </Book>
 <Book>
  <Title FontSize="10">Title-2</Title>
  <Price>150</Price>
 </Book>
</ListOfBooks>
```

21.1. Загрузка содержимого XML из файла

21.1.1. Вариант 1

Следующий код напечатает строку "Title-150Title-2150":

```
// Загрузка XML из файла
XmlDocument doc = new XmlDocument();
doc.Load("books.xml ");
// Показ содержимого XML
Console.WriteLine(doc.InnerText.ToString());
```

С помощью свойства `InnerXml` можно получить доступ к самому XML-коду документа:

```
// Напечатает XML-код документа
Console.WriteLine(doc.InnerXml.ToString());
```

21.1.2. Вариант 2

Для загрузки содержимого XML из файла можно использовать класс `XmlTextReader` (листинг 21.1).

Листинг 21.1. Загрузка содержимого XML из файла

```
using System;
using System.IO;
using System.Xml;

static void Main(string[] args)
{
  FileStream fs = new FileStream("Sample.xml", FileMode.Open);
  XmlTextReader xmlreader = new XmlTextReader(fs);
  while (xmlreader.Read())
  {
    Console.WriteLine("{0,-10}{1,-10}{2,-10}",
            xmlreader.NodeType.ToString(),
            xmlreader.Name,xmlreader.Value);
  }
  xmlreader.Close();
  fs.Close();
}
```

21.2. Загрузка содержимого XML из URL

Класс `XmlTextReader` умеет загружать XML из заданного адреса (листинг 21.2).

Листинг 21.2. Загрузка содержимого XML по URL

```
using System;
using System.Xml;

static void Main(string[] args)
{
  XmlTextReader xmlreader =
          new XmlTextReader("http://localhost/sample.xml");
  while (xmlreader.Read())
  {
     Console.WriteLine("{0,-10}{1,-10}{2,-10}",
            xmlreader.NodeType.ToString(),
```

```
            xmlreader.Name,xmlreader.Value);
  }
  xmlreader.Close();
}
```

21.3. Загрузка содержимого XML из строки

Класс `XmlTextReader` умеет загружать XML из строки (листинг 21.3).

Листинг 21.3. Загрузка содержимого XML из строки

```
using System;
using System.Xml;
using System.IO;
using System.Text;

static void Main(string[] args)
{
  string xmlData =
   "<?xml version='1.0' encoding='utf-8' ?><a><b>Some data</b></a>";
  UTF8Encoding utf8 = new UTF8Encoding();
  byte[] byteData = utf8.GetBytes(xmlData);
  MemoryStream ms = new MemoryStream(byteData);
  XmlTextReader xmlreader = new XmlTextReader(ms);
  while (xmlreader.Read())
  {
   Console.WriteLine("{0,-20}{1,-10}{2,-10}",
     xmlreader.NodeType.ToString(),
     xmlreader.Name,xmlreader.Value);
  }
  xmlreader.Close();
  ms.Close();
}
```

21.4. Обход всех элементов XML-файла

Обход всех элементов XML-файла показан в листинге 21.4.

Листинг 21.4. Обход всех элементов XML-файла

```
// Загрузка XML из файла
System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
```

```
doc.Load("books.xml");
System.Xml.XmlNode root = doc.DocumentElement;
// Напечатает "doc.DocumentElement=ListOfBooks"
Console.WriteLine("doc.DocumentElement={0}", root.LocalName);
foreach(System.Xml.XmlNode childnode in root.ChildNodes)
{
  // Напечатает сначала строку "Title-150", затем "Title-2150".
  // Эти строки являются слиянием текста двух
  // узлов корневого элемента
  Console.WriteLine(childnode.InnerText.ToString());
}
```

21.5. Поэлементное чтение XML-файла

Поэлементное чтение XML-файла с помощью `XmlTextReader` показано в листинге 21.5. Следует отметить, что будут отображены все типы элементов файла, включая разделительные пробелы (вывод приводится с сокращениями):

```
XmlDeclaration
Whitespace
Text:Title-1
EndElement
Whitespace
Text:50
EndElement
```

Листинг 21.5. Поэлементное чтение XML

```
System.Xml.XmlTextReader reader =
             new System.Xml.XmlTextReader("books.xml");
while(reader.Read())
{
  if(reader.NodeType == System.Xml.XmlNodeType.Element)
  {
   reader.Read(); // Читаем содержимое узла
   Console.WriteLine("{0}:{1}", reader.NodeType, reader.Value);
  } else {
   Console.WriteLine("{0}", reader.NodeType);
  }
}
```

21.6. Чтение атрибутов

Чтение атрибутов осуществляется с помощью метода `GetAttribute()`, как показано в листинге 21.6. Проверка на тип узла необходима, т. к. иначе будут найдены не только открывающие элементы (`XmlNodeType.Element`), но и закрывающие (`XmlNodeType.EndElement`).

Листинг 21.6. Чтение атрибутов

```
System.Xml.XmlTextReader reader =
         new System.Xml.XmlTextReader("ex001.xml");
while(reader.Read())
{
  if (reader.NodeType == System.Xml.XmlNodeType.Element)
  {
    if (reader.Name.Equals("Title"))
    {
      Console.WriteLine("<{0}>", reader.GetAttribute("FontSize"));
    }
  }
}
reader.Close();
```

21.7. Чтение всех атрибутов

Чтение всех атрибутов производится с помощью метода `MoveToNextAttribute()`, как показано в листинге 21.7.

Листинг 21.7. Чтение всех атрибутов

```
System.Xml.XmlTextReader reader =
                new System.Xml.XmlTextReader("ex001.xml");
while(reader.Read())
{
  if (reader.NodeType == System.Xml.XmlNodeType.Element)
  {
    if (reader.HasAttributes)
    {
      while (reader.MoveToNextAttribute())
      {
      Console.WriteLine("{0} = {1}", reader.Name, reader.Value);
      }
```

```
    }
  }
}
reader.Close();
```

21.8. Запись атрибутов

Запись атрибутов выполняется до записи содержимого элемента (листинг 21.8).

Листинг 21.8. Запись атрибутов

```
System.Xml.XmlTextWriter xmlwriter =
            new System.Xml.XmlTextWriter("ex002.xml", null);
xmlwriter.WriteStartDocument();                  // заголовок XML
xmlwriter.WriteStartElement("ListOfBooks");      // корневой элемент
xmlwriter.WriteStartElement("Book"); // книга 1
xmlwriter.WriteStartAttribute("FontSize", null); // атрибут
xmlwriter.WriteString("8");
xmlwriter.WriteEndAttribute();
xmlwriter.WriteString("Title-1");                // значение элемента
xmlwriter.WriteEndElement();
xmlwriter.WriteEndElement();
xmlwriter.Close();
```

21.9. Запись данных в XML-файл

Запись данных производится с помощью класса `System.Xml.XmlTextWriter` (листинг 21.9).

Методы `WriteStartElement` и `WriteEndElement` работают по принципу стека: `WriteEndElement` записывает закрывающий элемент последнего открывающего элемента, записанного с помощью `WriteStartElement`.

Листинг 21.9. Запись данных в XML-файл

```
System.Xml.XmlTextWriter xmlwriter =
            new System.Xml.XmlTextWriter("ex002.xml", null);
xmlwriter.WriteStartDocument();              // заголовок XML
xmlwriter.WriteStartElement("ListOfBooks");  // корневой элемент
xmlwriter.WriteStartElement("Book");         // книга 1
xmlwriter.WriteString("Title-1");
xmlwriter.WriteEndElement();
xmlwriter.WriteStartElement("Book");         // книга 2
```

```
xmlwriter.WriteString("Title-2");
xmlwriter.WriteEndElement();
xmlwriter.WriteEndElement();
xmlwriter.Close();
```

Некоторые классы умеют выполнять запись своего содержимого самостоятельно:

```
System.Data.DataSet ds = new System.Data.DataSet();
...
ds.WriteXml("result.xml");
```

21.10. Запись комментариев в XML-файл

Запись комментария производится с помощью метода `WriteComment` класса `System.Xml.XmlTextWriter`. Для записи строк на русском языке необходимо указывать кодировку файла.

```
System.Xml.XmlTextWriter xmlwriter =
          new System.Xml.XmlTextWriter("ex002.xml",
          System.Text.Encoding.GetEncoding(1251));
xmlwriter.WriteStartElement("ListOfBooks");
xmlwriter.WriteComment("Строка комментария");
xmlwriter.WriteEndElement();
xmlwriter.Close();
```

21.11. Запись пространства имен и префиксов в XML-файл

Пространство имен элемента можно указывать при его записи:

```
// Запишет <ListOfBooks>
xmlwriter.WriteStartElement("ListOfBooks", null);
// Запишет <ListOfBooks xmlns="http://localhost/test">
xmlwriter.WriteStartElement("ListOfBooks", "http://localhost/test");
// Запишет <prefix:ListOfBooks xmlns:prefix="http://localhost/test">
xmlwriter.WriteStartElement("prefix",
                    "ListOfBooks", "http://localhost/test");
```

21.12. Запись в XML-файл со специальным форматированием

По умолчанию XML-файл записывается без форматирования и представляет собой одну длинную строку. Для повышения читаемости файла можно указывать символы форматирования.

```
System.Xml.XmlTextWriter xmlwriter =
           new System.Xml.XmlTextWriter("ex002.xml", null);
// Включить форматирование документа
xmlwriter.Formatting = System.Xml.Formatting.Indented;
// Для выделения уровня элемента использовать табуляцию
xmlwriter.IndentChar = '\t';
// Использовать один символ табуляции
xmlwriter.Indentation = 1;
```

Аналогично можно указать выравнивание с помощью, например, двух пробелов:

```
xmlwriter.IndentChar = ' ';
xmlwriter.Indentation = 2;
```

По умолчанию строки в XML-файл записываются с помощью двойных кавычек. Использование одиночных кавычек производится следующим образом:

```
xmlwriter.QuoteChar = '\''; // использовать строки
                            // с одиночными кавычками
```

21.13. Выборка из XML с помощью XPath

Кроме обычных запросов (листинг 21.10) можно использовать заранее скомпилированные запросы XPath (листинг 21.11).

Листинг 21.10. Простые запросы XPath

```
// Загружаем XML-документ
XmlDocument doc = new XmlDocument();
doc.Load("book.xml");
// Получаем корневой элемент
XmlNode root = doc.DocumentElement;
// Получаем нужный узел
XmlNode node =
      root.SelectSingleNode("ListOfBooks/Book[position()=1]");
```

Листинг 21.11. Запросы XPath

```
// Создание XPath-документа
System.Xml.XPath.XPathDocument xpdoc =
       new System.Xml.XPath. XPathDocument("books.xml");
System.Xml.XPath.XPathNavigator nav = xpdoc.CreateNavigator();

// Прямой запрос XPath
System.Xml.XPath.XPathNodeIterator iterator1 =
                    nav.Select("ListOfBooks/Book/Title");
```

```
while (iterator1.MoveNext())
  Console.WriteLine(iterator1.Current);

// Скомпилированный запрос XPath
System.Xml.XPath.XPathExpression expr =
                    nav.Compile("ListOfBooks/Book/Price");
System.Xml.XPath.XPathNodeIterator iterator2 = nav.Select(expr);
while (iterator2.MoveNext())
  Console.WriteLine(iterator2.Current);
```

21.14. Вычисление выражений с помощью XPath

С помощью XPath можно производить вычисления, например, вычислить сумму указанных элементов (листинг 21.12).

Листинг 21.12. Вычисление суммы элементов с помощью XPath

```
// Вычисляющий запрос с предварительной компиляцией
System.Xml.XPath.XPathExpression expr1 =
            nav.Compile("sum(ListOfBooks/Book/Price/text())");
Console.WriteLine(expr1.ReturnType);
if (expr1.ReturnType == XPathResultType.Number)
{
  double summ = (double) nav.Evaluate(expr1);
  Console.WriteLine(summ);
}

// Вычисляющий запрос без предварительной компиляции.
// Кроме выборки производится арифметическое вычисление
double summ1 =
    (double) nav.Evaluate("sum(ListOfBooks/Book/Price/text())*10");
Console.WriteLine(summ1);
```

21.15. Вычисление *min* и *max* с помощью XPath

Спецификация XPath 1.1 не поддерживает функции `min` и `max`. Вычислить их значения можно с помощью более сложного запроса.

```
// Вычисление min
System.Xml.XPath.XPathExpression expr = nav.Compile("/ListOfBooks
        /Book/Price[not(. > /ListOfBooks/Book/Price)]/text()");
```

```
System.Xml.XPath.XPathNodeIterator iterator2 = nav.Select(expr);
iterator2.MoveNext();
Console.WriteLine(iterator2.Current);
```

Вычисление максимума производится заменой знака > на знак < в выражении XPath.

21.16. Создание *XPathDocument* из строки

Самый простой способ:

```
System.IO.StringReader srxml = new System.IO.StringReader(str);
System.Xml.XPath.XPathDocument xpdoc =
          new System.Xml.XPath.XPathDocument(srxml);
```

21.17. XSL-трансформация

XSL-трансформация с помощью класса `System.Xml.Xsl.XslTransform` выполняется так:

```
System.Xml.Xsl.XslTransform xslt =
              new System.Xml.Xsl.XslTransform();
xslt.Load("books.xsl");
System.IO.StringWriter output = new System.IO.StringWriter();
xslt.Transform(nav, null, output);
Console.WriteLine(output);
```

Содержимое файла books.xsl:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:template match="/">
<html>
  <xsl:value-of select="ListOfBooks/Book/Title"/>
 </html>
 <hr/>
 <xsl:for-each select="ListOfBooks/Book">
  Title=<xsl:value-of select="./Title"/><br/>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

Результатом будет следующий текст:

```
<html>Title-1</html>
<hr />
```

```
Title=Title-1<br />
Title=Title-2<br />
```

При необходимости получить на выходе файл в XML-формате в XSL-файл нужно добавить строку:

```
<xsl:output method="xml" version="1.0" indent="yes"/>
```

21.18. XSL-трансформация с параметрами

Для передачи параметров необходимо, во-первых, создать экземпляр класса `Xml.Xsl.XsltArgumentList`, а во-вторых, указать параметры в XSL-файле. Таким образом, XSL-файл может быть таким (books.xsl):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
<xsl:param name='min_pos'>0</xsl:param>
<xsl:param name='max_pos'>0</xsl:param>

<xsl:template match="/">
<xsl:for-each select="//Book[position() &lt;= $max_pos
                             and position() &gt;= $min_pos]">
  <xsl:apply-templates select="Title"/>
</xsl:for-each>

</xsl:template>
</xsl:stylesheet>
```

Код трансформации выглядит следующим образом:

```
System.Xml.Xsl.XslTransform xslt =
            new System.Xml.Xsl.XslTransform();
xslt.Load("books.xsl");
System.IO.StringWriter output = new System.IO.StringWriter();
// Параметры преобразования
System.Xml.Xsl.XsltArgumentList xsltArgs =
            new System.Xml.Xsl.XsltArgumentList();
xsltArgs.AddParam("min_pos", "", 1); // выбираем с 1 по 5 позиции
xsltArgs.AddParam("max_pos", "", 5);
xslt.Transform(nav, xsltArgs, output);
Console.WriteLine(output);
```

Как видно, здесь с помощью параметров мы выбираем все записи `Book`, с минимальной по максимальную указанную позицию, что позволяет делать, например, разбивку на страницы.

21.19. Проверка файла с помощью XSD

С помощью XSD можно проверить схему XML-документа. Листинг 21.15 содержит пример проверки XML-файла, приведенного в листинге 21.13, с помощью схемы из листинга 21.14.

Листинг 21.13. Пример XML-файла

```
<?xml version="1.0" encoding="utf-8" ?>
<address>
  <name>John Smith</name>
  <street>109 Abbey Close</street>
  <city>Hayes</city>
  <country> UK</country>
</address>
```

Листинг 21.14. Пример XSD-файла

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="address">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="name"    type="xs:string"/>
    <xs:element name="street"  type="xs:string"/>
    <xs:element name="city"    type="xs:string"/>
    <xs:element name="country" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Листинг 21.15. Программа проверки схемы с помощью XSD

```
using System;
using System.Collections;
using System.Data;
using System.IO;
using System.Xml;
using System.Xml.Schema;
using System.Text;

namespace XSD
{
  public class XMLValidator
```

```
{
  string xmlFileName;
  string xsdFileName;

  public XMLValidator(string xmlFileName, string xsdFileName)
  {
    this.xmlFileName = xmlFileName;
    this.xsdFileName = xsdFileName;
  }

  // Сообщение об ошибке
  static string errorMessage = string.Empty;
  public string ErrorMessage
  {
    get { return errorMessage; }
  }

  // Обработчик проверки
  public static void ValidationHandler(
                   object sender,  ValidationEventArgs args)
  {
    errorMessage += args.Message + "\r\n";

  }

  public void Validate()
  {
    try
    {
      // Схема
      XmlTextReader schemaReader = new XmlTextReader(xsdFileName);
      XmlSchemaCollection schema = new XmlSchemaCollection();
      schema.Add(null, schemaReader);

      // Валидатор
      XmlTextReader xmlReader = new XmlTextReader(xmlFileName);
      XmlValidatingReader vr = new XmlValidatingReader(xmlReader);
      vr.Schemas.Add(schema);
      vr.ValidationType = ValidationType.Schema;
      vr.ValidationEventHandler +=
                 new ValidationEventHandler(ValidationHandler);

      // Проверка
      while(vr.Read());
```

```
        vr.Close();
      }
      catch (Exception e)
      {
        errorMessage += e.Message + "\r\n";
      }
    }
  }

  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      XMLValidator v = new XMLValidator("test.xml", "test.xsd");
      v.Validate();
      if (v.ErrorMessage.Length > 0)
        Console.WriteLine(v.ErrorMessage);
      else
        Console.WriteLine("OK");
    }
  }
}
```

21.20. В чем разница между *InnerXml*, *OuterXml* и *InnerText*?

Следующий пример показывает разницу между этими свойствами:

```
System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
doc.LoadXml("<book genre='novel' ISBN='1-861001-57-5'>" +
    "<title>Pride And Prejudice</title>" +
    "</book>");

System.Xml.XmlNode root = doc.DocumentElement;

// OuterXml включает атрибуты текущего узла.
// Напечатает <book genre="novel" ISBN="1-861001-57-5">
// <title>Pride And Prejudice</title></book>
Console.WriteLine(root.OuterXml);

// InnerXml не включает атрибуты текущего узла.
// Напечатает <title>Pride And Prejudice</title>
Console.WriteLine(root.InnerXml);
```

```
// InnerText содержит текст узла.
// Напечатает Pride And Prejudice
Console.WriteLine(root.InnerText);
```

21.21. В чем разница между *Load* и *LoadXml* класса *XmlDocument*?

Метод `Load()` загружает XML-документ из файла или указанного URL, а `LoadXml` — из строки. При этом есть существенная разница при разборе документа: метод `Load()` умеет работать с документами, имеющими метатеги, например:

```
<?xml version="..." charset="..." ?>
```

Попытка загрузить документ с метатегами с помощью `LoadXml()` вызовет исключение. Таким образом, если вручную считать XML-документ в строку и затем загружать его с помощью метода `LoadXml()`, то об отсечении метатегов необходимо заботиться самостоятельно.

21.22. Как добавить *XmlNode* из одного документа в другой?

Метод `XmlNode.AppendChild` позволяет добавлять узлы только в тот документ, которому принадлежит узел. Для копирования `XmlNode` можно применить следующий код:

```
// destParent и sourceParent принадлежат разным XmlDocuments
public static void TransferChildren(XmlDocument destDoc,
        XmlNode destParent, XmlNode sourceParent)
{
  // Создаем временный элемент
  XmlElement tempElem = destDoc.CreateElement("Temp");
  // Переносим содержимое
  tempElem.InnerXml = sourceParent.InnerXml;
  // Переносим потомков
  foreach(XmlNode node in tempElem.ChildNodes)
      destParent.AppendChild(node);
}
```

21.23. Аналог простого ini-файла для хранения настроек

В одном из FAQ предлагается довольно простой способ хранения настроек в XML-файле. Для доступа к свойствам используется класс `DataSet`:

```
System.Data.DataSet ds = new System.Data.DataSet();
ds.ReadXml("config.ini", XmlReadMode.Auto);

System.Data.DataTable table = ds.Tables["settings"];
System.Data.DataRow row = table.Rows[0];

string baseFolder = (string)row["base_path"];
string updateFolder = (string)row["update_path"];
string outputFileName = (string)row["output_file"];
```

Сам файл может выглядеть следующим образом:

```
<?xml version="1.0" standalone="yes" ?>
 <config>
 <settings>
      <base_path>folder1</base_path>
      <update_path>folder2</update_path>
      <output_file>patch.bin</output_file>
   </settings>
 </config>
```

После изменения настроек можно сохранить файл:

```
row["base_path"] = "test";
ds.WriteXml("config+.ini");
```

21.24. Работа с ini-файлами

В одной из статей MSDN (автор Ральф Вестфол (Ralf Westphal)) предлагается следующий способ работы с ini-файлами. Сначала ini-файл открывается как обычный текстовый файл. Затем файл переводится в XML-формат. Например:

```
[MySettings]
Value1=1
Type=2
Auto=true
```

Такой файл будет преобразован в следующую структуру:

```
<xml>
  <section name=" MySettings ">
    <item name="Value1" value="1"/>
    <item name="Type" value="2"/>
    <item name="Auto" value="true"/>
  </section>
</xml>
```

С XML-файлом работа производится стандартными средствами C#. При необходимости сохранить настройки этот файл снова записывается в виде ini-файла.

ГЛАВА 22

Windows Forms

22.1. Сворачивание программы в трей

Для добавления возможности сворачивания программы в трей необходимо выполнить следующую последовательность действий:

1. Добавить на форму компонент **NotifyIcon** и установить его свойства:
   - свойство `Text` задает текст, который будет отображаться при подведении указателя мыши к значку в трее;
   - свойство `Icon` задает сам значок, отображаемый в трее.
2. Добавить обработчик события формы `Resize`, скрывающий приложение из панели задач при минимизации:

```
private void Form1_Resize(object sender, System.EventArgs e)
{
  if (FormWindowState.Minimized == WindowState)
    Hide();
}
```

3. Добавить обработчик события `NotifyIcon.DoubleClick`, восстанавливающий приложение по двойному щелчку на значке:

```
private void notifyIcon1_DoubleClick(object sender,
                                     System.EventArgs e)
{
  Show();
  WindowState = FormWindowState.Normal;
}
```

4. При необходимости, добавить компонент `ContextMenu` и привязать его к значку в трее с помощью свойства `NotifyIcon.ContextMenu`.

22.2. Как сделать форму, не видимую в панели задач?

Для того чтобы форма не показывалась в панели задач, необходимо установить свойство `ShowInTaskbar` в значение `false`.

22.3. Как скрыть главную форму при старте приложения?

Установите свойства формы: `ShowInTaskbar` равным `true`, а `WindowState` равным `FormWindowState.Minimize`. Для показа формы (например, по срабатыванию таймера) необходимо восстановить значения этих свойств:

```
private void timer1_Elapsed(object sender,
                            System.Timers.ElapsedEventArgs e)
{
  this.ShowInTaskbar = true;
  this.WindowState = FormWindowState.Normal;
}
```

22.4. Получение всех цветов в системе

Получить все цвета в системе можно с использованием отражения *(см. также главу 4)*:

```
System.Reflection.PropertyInfo[] PropArray =
       System.Drawing.Color.Empty.GetType().GetProperties();
foreach(System.Reflection.PropertyInfo prop in PropArray)
{
  // Фильтруем методы, оставляем только цвета
  if(prop.PropertyType == typeof(System.Drawing.Color))
  listBox1.Items.Add(prop.Name);
}
```

Или с помощью методов класса `Enum` *(см. также разд. 8.1)*:

```
string[] colors= Enum.GetNames(typeof(System.Drawing.KnownColor));
foreach (string color in colors)
{
  if(Color.FromKnownColor(
   (KnownColor)Enum.Parse(typeof(KnownColor), color)).IsSystemColor)
  {
   // Системные цвета
   listBox1.Items.Add(color);
  }
```

```
  else
  {
    // Остальные цвета
  }
}
```

22.5. Получение положения курсора мыши

Класс `System.Windows.Forms.Cursor` отвечает за курсор мыши, метод `Position` возвращает координаты мыши:

```
private void Form1_MouseMove(object sender,
                      System.Windows.Forms.MouseEventArgs e)
{
  // Выводит "{X=200,Y=100}"
  label1.Text = System.Windows.Forms.Cursor.Position.ToString();
  // Выводим x и y
  label2.Text = string.Format("X={0} Y={0}",
     System.Windows.Forms.Cursor.Position.X,
     System.Windows.Forms.Cursor.Position.Y
     );
}
```

22.6. Как отлавливать все исключения?

Самый очевидный способ — ловить исключения в функции `Main`:

```
[STAThread]
static void Main()
{
  try
  {
    Application.Run(new Form1());
  }
  catch(Exception e)
  {
  }
}
```

Еще один способ — использование обработчика `ThreadException`:

```
[STAThread]
static void Main()
{
  CustomExceptionHandler eh = new CustomExceptionHandler();
```

```
  Application.ThreadException += new
    System.Threading.ThreadExceptionEventHandler(
                                      eh.OnThreadException);
  Application.Run(new Form1());
 }

internal class CustomExceptionHandler
{
   // Обработчик исключения
   public void OnThreadException(object sender,
            System.Threading.ThreadExceptionEventArgs t)
  {
     MessageBox.Show(t.Exception.Message);
   }
}
```

См. также разд. 5.7 и 24.5.

22.7. Как сделать обработчик для сообщений Win32?

Перехватить сообщения Win32 можно с помощью интерфейса `System.Windows.Forms.IMessageFilter` (листинг 22.1). К сожалению, сообщения приходится обрабатывать по коду. Параметры сообщения доступны с помощью свойств `m.LParam` и `m.WParam`. Второй вариант — использовать перекрытие метода `WndProc` (листинг 22.2).

Листинг 22.1. Обработчик сообщений Win32

```
// Класс фильтра
public class MyMessageFilter : System.Windows.Forms.IMessageFilter
{
   public bool PreFilterMessage(ref Message m)
   {
      // Нажатие левой кнопки мыши
      if (m.Msg == 513)
      {
         MessageBox.Show("WM_LBUTTONDOWN is: " + m.Msg);
         return true;
      }
      return false;
   }
}
```

```
// Объект фильтра
static private MyMessageFilter msgFliter = new MyMessageFilter();

[STAThread]
static void Main()
{
  // Регистрация объекта фильтра
  Application.AddMessageFilter(msgFliter);
  Application.Run(new Form1());
}
```

Листинг 22.2. Метод `WndProc`

```
protected override void WndProc(ref Message m)
{
  lbLog.Items.Add(String.Format("Msg={0} L={1} W={2}",
          m.Msg.ToString(), m.LParam.ToString(),
          m.WParam.ToString()));
  lbLog.SelectedIndex = lbLog.Items.Count-1;
  // Обработка сообщения с кодом m.Msg
  base.WndProc(ref m);
}
```

22.8. Есть ли в C# аналог *ProcessMessages* из Delphi?

Есть. Метод называется `Application.DoEvents()`.

22.9. Проблемы с *ImageList* при использовании *EnableVisualStyles*

Код для исправления проблемы с `ImageList` при вызове `EnableVisualStyles` выглядит следующим образом (автор Мартин Робинс (Martin Robins)):

```
public virtual void Main()
{
  Application.EnableVisualStyles();
  // Вызов DoEvents после метода, указанного выше, решает проблему
  Application.DoEvents();
  Application.Run(new Form1());
}
```

22.10. Как поменять фон MDI-окон?

Изменение фона MDI-окон:

```
private void Form_Load(object sender, System.EventArgs e)
{
  foreach (Control ctl in this.Controls)
  {
     if (ctl is MdiClient) // found it
     {
        ctl.BackColor = Color.Yellow;
        break;
     }
  }
}
```

22.11. Определение разрешения экрана

Для определения разрешения экрана можно воспользоваться свойством `System.Windows.Forms.Screen.PrimaryScreen.Bounds`.

22.12. Определение рабочей области экрана без системного трея

Метод `Screen.GetWorkingArea(control)` позволяет получить рабочую область экрана без области системного трея.

Например, для открытия формы в углу системного трея можно использовать следующий код (установив предварительно свойство формы `StartPosition` в значение `Manual`):

```
this.SetBounds(Screen.GetWorkingArea(this).Width  – this.Width ,
               Screen.GetWorkingArea(this).Height – this.Height,
               this.Width, this.Height);
```

22.13. Использование системного буфера обмена

Для работы с буфером обмена предназначены методы `SetDataObject` и `GetDataObject` класса `System.Windows.Forms.Clipboard` (листинг 22.3).

Листинг 22.3. Использование буфера обмена

```
private void button1_Click(object sender, System.EventArgs e)
{
  string text = "Текст";
  Clipboard.SetDataObject(text);

  IDataObject data = Clipboard.GetDataObject();
  if(data.GetDataPresent(DataFormats.Text))
  {
    textBox1.Text = (String)data.GetData(DataFormats.Text);
  }
}
```

22.14. Обработка изменения системных настроек (например, разрешения экрана)

Сделать обработчик изменения системных настроек можно с помощью статических событий класса `Microsoft.Win32.SystemEvents` (листинг 22.4).

Листинг 22.4. Обработка изменения системных настроек

```
private void Form1_Load(object sender, System.EventArgs e)
{
  Microsoft.Win32.SystemEvents.DisplaySettingsChanged +=
             new System.EventHandler(displaySettingsChanged);
  Microsoft.Win32.SystemEvents.UserPreferenceChanged +=
      new Microsoft.Win32.UserPreferenceChangedEventHandler(
                  userPreferencesChanged);
}

private void displaySettingsChanged(object sender, EventArgs e)
{
  MessageBox.Show(e.ToString());
}

private void userPreferencesChanged(object sender,
         Microsoft.Win32.UserPreferenceChangedEventArgs e)
{
  switch(e.Category)
  {
    case Microsoft.Win32.UserPreferenceCategory.Locale:
```

```
        MessageBox.Show("Changed locale");
        break;
    default:
        MessageBox.Show(e.Category.ToString());
        break;
  }
}
```

22.15. Показ HTML на форме

Для показа HTML следует воспользоваться компонентом **WebBrowser**. Для его использования нужно щелкнуть правой кнопкой мыши на панели компонентов, выбрать меню **Add/Remove Items** и добавить COM-объект **Microsoft WebBrowser**. После добавления этого компонента на форму его следует инициализировать следующим образом:

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

  object a = 0;
  object b = 0;
  object c = 0;
  object d = 0;
  axWebBrowser1.Navigate("www.microsoft.com",
                         ref a, ref b, ref c, ref d);
}
```

22.16. Отображение подсказки из справочных файлов

Отображение подсказки из chm-файла осуществляется с помощью методов класса `System.Windows.Forms.Help` (листинг 22.5).

Листинг 22.5. Отображение подсказки

```
// Простой запуск подсказки из файла
System.Windows.Forms.Help.ShowHelp(this,"TestHelp.chm");
// Отображение нужной темы из файла
System.Windows.Forms.Help.ShowHelp(this,"TestHelp.chm",
                         "topic1.htm");
// Отображение индекса
System.Windows.Forms.Help.ShowHelpIndex(this,"TestHelp.chm");
```

```
// Отображение всплывающей подсказки
System.Windows.Forms.Help.ShowPopup(this,"This is a Help pop-up",
             new Point(100,100));
```

22.17. Рисование без обработки события *Paint*

Обычно для рисования необходимо обработать событие `Paint`. Однако можно рисовать и другим способом — с помощью объекта `hwnd` (листинг 22.6). Правда, в этом случае заботиться об обновлении изображения придется вручную (например, при возврате после переключения на другое окно по нажатию комбинации клавиш <Atl>+<Tab> изображение исчезнет).

Листинг 22.6. Рисование через `hwnd`

```
Graphics g = Graphics.FromHwnd(this.Handle);
SolidBrush brush = new SolidBrush(Color.Red);
Rectangle rectangle = new Rectangle(25, 25, 100, 100);
g.FillRectangle(brush, rectangle);
```

22.18. Как нарисовать точку?

Самый простой способ нарисовать точку — это нарисовать маленький кружок. Например:

```
e.Graphics.FillRectangle(
             new SolidBrush(Color.Black), 10, 10, 1, 1);
```

22.19. Проверка на попадание в область

Метод `IsVisible` используется для проверки на попадание точки в область (листинг 22.7).

Листинг 22.7. Проверка на попадание точки в область

```
public System.Drawing.Drawing2D.GraphicsPath path;

private void Form1_Load(object sender, System.EventArgs e)
{
  // Создаем некоторую область
  path = new System.Drawing.Drawing2D.GraphicsPath();
  path.AddLine(0,0,100,100);
  path.AddLine(100,100,50,20);
  path.AddLine(50,20,0,0);
}
```

```
private void panel1_Paint(object sender,
                System.Windows.Forms.PaintEventArgs e)
{
  // Рисуем
  e.Graphics.DrawPath(new Pen(Color.Red), path);
}

private void panel1_MouseMove(object sender,
                     System.Windows.Forms.MouseEventArgs e)
{
  // Проверяем, попадала ли точка в область
  Region region = new Region(path);
  if (region.IsVisible(e.X, e.Y))
  label1.Text = "IN";
  else
  label1.Text = "OUT";
}
```

22.20. Вывод на консоль из приложения Windows Form

Вывод на консоль из приложения Windows Form требуется, например, если программа должна запускаться с параметрами. При запуске без параметров программа не показывает форму, а выводит на консоль правила запуска и информацию о параметрах. Специальный класс `ConsoleAPI` (листинг 22.9) позволяет реализовать такую функциональность (листинг 22.8).

Листинг 22.8. Вывоз на консоль из приложения Windows Form

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

namespace WinFormConsole
{
  public class Form1 : System.Windows.Forms.Form
  {
    private System.Windows.Forms.Label label1;

    static string argValue;
```

```
    [STAThread]
    static void Main(string [] args)
    {
      if (args.Length == 0)
      {
        using (ConsoleAPI c = new ConsoleAPI())
        {
          Console.WriteLine("Эту программу надо
                             запускать с параметрами!");
          Console.ReadLine();
        }
        return;
      }

      argValue = args[0];

      Application.Run(new Form1());
    }

    private void Form1_Load(object sender, System.EventArgs e)
    {
      label1.Text = argValue;
    }
  }
}
```

Листинг 22.9. Класс `WinFormConsole`

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

namespace WinFormConsole
{
  public class ConsoleAPI : IDisposable
  {
    private static IntPtr conOut;
    private static IntPtr oldOut;

    public ConsoleAPI()
    {
      if (oldOut == IntPtr.Zero)
        oldOut = GetStdHandle( -11 );
```

```
      if (!AllocConsole())
        throw new Exception("AllocConsole");

      conOut = CreateFile( "CONOUT$", 0x40000000, 2,
                           IntPtr.Zero, 3, 0, IntPtr.Zero );

      if (!SetStdHandle(-11, conOut))
        throw new Exception("SetStdHandle");

  Stream standard = Console.OpenStandardOutput();
  StreamWriter writer = new StreamWriter(standard);
  writer.AutoFlush = true;
  Console.SetOut(writer);
  Console.SetError(writer);
}

    public void Dispose()
    {
      if (! CloseHandle(conOut))
        throw new Exception("CloseHandle");
      conOut = IntPtr.Zero;
      if (! FreeConsole())
        throw new Exception("FreeConsole");
      if (!SetStdHandle(-11, oldOut))
        throw new Exception("SetStdHandle");
    }

    [DllImport("kernel32.dll", SetLastError=true)]
    protected static extern bool AllocConsole();

    [DllImport("kernel32.dll", SetLastError=false)]
    protected static extern bool FreeConsole();

    [DllImport("kernel32.dll", SetLastError=true)]
    protected static extern IntPtr GetStdHandle(int nStdHandle);

    [DllImport("kernel32.dll", SetLastError=true)]
    protected static extern bool SetStdHandle(
                    int nStdHandle, IntPtr hConsoleOutput );

    [DllImport("kernel32.dll", CharSet=CharSet.Auto,
                                  SetLastError=true)]
    protected static extern IntPtr CreateFile(
      string  fileName,
      int   desiredAccess,
```

```
      int   shareMode,
      IntPtr  securityAttributes,
      int   creationDisposition,
      int   flagsAndAttributes,
      IntPtr  templateFile );

    [DllImport("kernel32.dll", ExactSpelling=true,
                                        SetLastError=true)]
    protected static extern bool CloseHandle(IntPtr handle);
  }
}
```

22.21. Использование автозавершения в текстовом поле

Следующий класс позволяет включить режим автозавершения в поле ввода (как в поле ввода после выбора команды **Пуск | Выполнить** (Start | Run) или в Internet Explorer). Для включения режима автозавершения достаточно вызвать метод `EnableAutoComplete` (листинг 22.11) класса `AutoCompleteText` (листинг 22.10).

Листинг 22.10. Класс управления автозавершением текста

```
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace AutoCompleteText
{
  public class AutoCompleteText
  {
    [DllImport("SHLWAPI", CharSet=CharSet.Auto, SetLastError=true)]
    static extern int SHAutoComplete(
                  IntPtr hwndEdit, SHAutoCompleteFlags dwFlags);

    [Flags]
    enum SHAutoCompleteFlags: uint
    {
      /// <summary>
      /// Currently (FileSystem | UrlAll)
      /// </summary>
      Default=0x00000000,
```

```
/// <summary>
/// This includes the File System as well as the rest
/// of the shell (Desktop\My Computer\Control Panel\)
/// </summary>
FileSystem=0x00000001,

/// <summary>
/// URLs in the User's History and URLs in the User's
/// Recently Used list.
/// </summary>
UrlAll=(UrlHistory | UrlMRU),

/// <summary>
/// URLs in the User's History
/// </summary>
UrlHistory=0x00000002,

/// <summary>
/// URLs in the User's Recently Used list.
/// </summary>
UrlMRU=0x00000004,

/// <summary>
///  Use the tab to move thru the autocomplete
///  possibilities instead of to the next dialog/window
///  control.
/// </summary>
UseTab=0x00000008,

/// <summary>
/// Don't AutoComplete non-File System items.
/// </summary>
FileSysOnly=0x00000010,

/// <summary>
/// Ignore the registry default and force the feature on.
/// </summary>
AutoSuggestForceOn=0x10000000,

/// <summary>
/// Ignore the registry default and force the feature off.
/// </summary>
AutoSuggestForceOff=0x20000000,
```

```
      /// <summary>
      /// Ignore the registry default and force the feature on.
      /// (Also known as AutoComplete)
      /// </summary>
      AutoAppendForceOn=0x40000000,

      /// <summary>
      /// Ignore the registry default and force the feature off.
      /// (Also known as AutoComplete)
      /// </summary>
      AutoAppendForceOff=0x80000000
    }

    public static void EnableAutoComplete(IntPtr handle,
      bool autoCompleteUrls, bool autoCompleteFiles)
    {
      if (autoCompleteUrls || autoCompleteFiles)
      {
        Application.OleRequired();

        SHAutoCompleteFlags flags =
          SHAutoCompleteFlags.AutoSuggestForceOn |
          SHAutoCompleteFlags.AutoAppendForceOn;

        if (autoCompleteUrls ) flags |= SHAutoCompleteFlags.UrlAll;
        if (autoCompleteFiles) flags |=
                        SHAutoCompleteFlags.FileSystem;
        SHAutoComplete(handle, flags);
      }
    }
  }
}
```

Листинг 22.11. Пример использования класса `AutoCompleteText`

```
private void button1_Click(object sender, System.EventArgs e)
{
  AutoCompleteText.EnableAutoComplete(textBox1.Handle, true, true);
}
```

ГЛАВА 23

Базы данных

23.1. Что следует использовать для закрытия соединения — *Close* или *Dispose*?

Отличие `Close` от `Dispose` следующее. Метод `Close` закрывает соединение, но оставляет доступным сам объект, т. е. можно работать со свойствами соединения или открывать его заново. Метод `Dispose` уничтожает объект без возможности его дальнейшего использования. Вызов `Dispose`, однако, не означает, что соединение будет удалено из пула соединений.

23.2. Работа с файлами MS Access

Пример работы с файлами MS Access продемонстрирован в листинге 23.1.

Листинг 23.1. Работа с файлом MS Access

```
using System;
using System.Data.OleDb;

namespace OldDb
{
  class OleDbTest
  {
    public static void Main()
    {
      // Создаем соединение
      OleDbConnection aConnection =
           new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
                                        Data Source=c:\\db1.mdb");
```

```
      // Создаем объект для выполнения SQL
      OleDbCommand aCommand = new OleDbCommand("select *
                                  from emp_test", aConnection);
      try
      {
        aConnection.Open();
        // Создаем DataReader
        OleDbDataReader aReader = aCommand.ExecuteReader();
        Console.WriteLine("This is the returned data from
                                           emp_test table");

        // Читаем данные
        while(aReader.Read())
        {
          Console.WriteLine(aReader.GetInt32(0).ToString());
        }
        // Закрываем DataReader
        aReader.Close();
        // Закрываем соединение
        aConnection.Close();
      }
      // Обработка ошибок
      catch(OleDbException e)
      {
         Console.WriteLine("Error: {0}", e.Errors[0].Message);
      }
    }
  }
}
```

23.3. Получение индекса объекта после добавления его в таблицу

Пусть имеется некий класс, который будет использоваться в бизнес-логике приложения.

```
class Test
{
  int     ID;
  string  Name;
}
```

Таблица, в которой хранятся эти объекты, имеет автоинкрементное поле `ID`. При чтении списка таких объектов и их удалении проблем не возникает, а вот

добавление часто вызывает вопрос: "Как узнать следующее значение ID?" Один из абсолютно неправильных вариантов, который я часто встречал, — до или после добавления новой записи выполнять запрос SELECT TOP 1 FROM MYTABLE ORDER BY ID DESC. Действительно, если в таблицу добавляется много записей разными пользователями, то вероятность "поймать" нужный индекс практически равна нулю. Еще один вариант — после каждого добавления перечитывать список объектов заново. Это будет работать, но чем больше объектов в списке, тем медленнее.

Правильный подход — добавление или в хранимую процедуру, или в SQL-запрос кода, возвращающего значение SCOPE_IDENTITY(). Это специальная функция MS SQL, которая возвращает значение автоинкрементного столбца, записанного сервером в рамках текущего пакета.

Следует помнить, что в MS SQL существует еще одна аналогичная функция — @@IDENTITY. Пользоваться ею надо осторожно. В отличие от SCOPE_IDENTITY(), эта функция возвращает значение автоинкрементного столбца, записанного сервером в рамках текущей сессии. Это означает, что если выполняется простой запрос INSERT, то эти функции будут работать одинаково, но при вставке сработает триггер, который выполнит свой INSERT, и @@IDENTITY вернет значение, вставленное триггером, т. к. вставка будет выполнена все еще в рамках одной сессии. Поэтому предпочтительнее пользоваться функцией SCOPE_IDENTITY().

23.4. Можно ли сделать *GroupBy* в *DataSet*?

Можно, но кода нужно довольно много:
см. **support.microsoft.com/kb/326145/EN-US**.

23.5. Перечисление доступных SQL-серверов

С помощью функции NetServerEnum можно получить список доступных SQL-серверов (листинг 23.2).

Листинг 23.2. Перечисление доступных SQL-серверов

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

namespace SQLServersEnum
{
  internal class Class1
```

```
{
  [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, uint ServerType,
       [MarshalAs(UnmanagedType.LPWStr)] string Domain,
       int ResumeHandle);

  [DllImport("netapi32.dll", EntryPoint="NetApiBufferFree")]
  public static extern NERR NetApiBufferFree (IntPtr Buffer);

  // Тип сервера
  const uint SV_TYPE_SQLSERVER = 0x00000004;

  [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;
  }


  // Операционная система
  public enum PLATFORM_ID : uint
  {
    PLATFORM_ID_DOS = 300,
    PLATFORM_ID_OS2 = 400,
    PLATFORM_ID_NT = 500,
    PLATFORM_ID_OSF = 600,
    PLATFORM_ID_VMS = 700,
  }

  // 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,
}

// 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_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);
    }

    [STAThread]
    static void Main()
    {
      ArrayList list = GetSQLServerList();
      foreach (string name in list)
      {
        Console.WriteLine(name);
      }
    }
  }
}
```

23.6. Создание расширений MS SQL 2005

SQL 2005 предоставляет возможность создания расширений с помощью C# 2.0 (автор Джаспер Смит (Jasper Smith), **www.sqldbatips.com /showarticle.asp?ID=22**). Пример кода такого модуля приведен в листинге 23.3. Код компилируется с помощью команды

```
csc /target:library c:\HelloWorld.cs /r:"C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Binn\sqlaccess.dll"
```

Полученную сборку нужно загрузить в MS SQL, используя команду:

```
create assembly HelloWorld from 'c:\HelloWorld.dll' with permission_set =
safe
```

Теперь можно создать процедуру с помощью команды:

```
create procedure HelloWorldas external name HelloWorld.SQLCLR.HelloWorld
```

Пример выполнения:

```
exec HelloWorld
[Results]
Hello World, from SQLCLR!!
```

Пример создания триггера **www.15seconds.com/issue/041006.htm** (автор Тури Тангаратинам (Thiru Thangarathinam)).

Листинг 23.3. Модуль расширения для MS SQL 2005 (C# 2.0)

```
using System.Data.SqlServer;
using System.Data.SqlTypes;

public class SQLCLR{

   public static void HelloWorld()
   {
      // Получаем объект SQL-контекста
      SqlPipe sqlP = SqlContext.GetPipe();
      // и просто передаем строку
      sqlP.Send("Hello World, from SQLCLR!!");
   }
}
```

ГЛАВА 24

ASP.NET

24.1. Преобразование относительного пути в абсолютный

Операция преобразования относительного пути в абсолютный требуется, например, при необходимости загрузить конфигурационный файл. Выполнить преобразование можно с помощью метода `Server.MapPath`:

```
XmlDocument doc = new XmlDocument();
doc.Load(Server.MapPath("config.xml"));
```

Корневой каталог можно получить с помощью следующих запросов:

```
<%= Server.MapPath("/")%>
```

или

```
<%= Server.MapPath("")%>
```

И тот и другой запрос вернет строку `"c:\inetpub\wwwroot"` или путь виртуального каталога.

24.2. Управление буферизацией страниц

Для того чтобы страница отсылалась клиенту только после окончания генерации, а не по частям, можно установить `Response.BufferOutput=true`. Недостаток этого режима — пользователь не может прервать загрузку страницы.

24.3. Управление кэшированием страниц

Отмена кэширования страницы:

```
// Не сохранять
Response.Cache.SetNoStore();
```

```
// Не сохранять в истории браузера
Response.Cache.SetAllowResponseInBrowserHistory(false);
// Не кэшировать на сервере
Response.Cache.SetNoServerCaching();
```

Для запрещения кэширования страниц в заголовке страницы указывается срок действительности страницы равный нулю:

```
<% Response.Expires = 0 %>
```

Для управления кэшированием на прокси-серверах указывается заголовок:

```
<% Response.CacheControl = "Public" %>
```

Значением по умолчанию является строка `"Private"`, которая запрещает кэширование страниц прокси-серверами.

24.4. Перенаправление на другую страницу

Для перенаправления на другую страницу можно использовать метод `Server.Transfer` или `Response.Redirect`.

24.5. Чем метод *Server.Transfer* отличается от метода *Response.Redirect*?

Для перенаправления запроса существуют два метода: `Response.Redirect` и `Server.Transfer`. Фактически они выполняют одну и ту же функцию, но между ними есть существенная разница.

При использовании метода `Response.Redirect` сервер направляет обозревателю (клиенту) HTTP-ответ, в котором содержится расположение нового URL. Клиент выходит из очереди запросов данного сервера и отправляет новый HTTP-запрос по указанному адресу. Сервер добавляет этот запрос в очередь запросов вместе с запросами других клиентов, поступившими за это время. Таким образом, перенаправление с помощью `Response.Redirect` требует двойного обмена, что повышает нагрузку сервера.

Метод `Server.Transfer` отправляет запросы из одного исполняемого ASP-файла в другой. Во время передачи первоначально запрошенный ASP-файл сразу же прекращает выполнение, но не очищает выходной буфер. Таким образом, происходит "подмена" содержимого непосредственно на стороне сервера. Кроме минимизации количества запросов, это означает, что новый файл имеет доступ к тому же набору внутренних объектов (`Request`, `Response`, `Server`, `Session` и `Application`), что и первоначально запрошенный файл. Кроме того, использование метода `Server.Transfer` допускается для обмена ин-

формацией между ASP-файлами, расположенными в различных приложениях. Тем не менее при передаче в ASP-файл, расположенный в другом приложении, этот файл будет вести себя так, как если бы он был частью приложения, начавшего передачу. Это означает, что указанный файл будет иметь доступ только к тем переменным, область определения которых задана для исходного приложения, а не того приложения, в котором этот файл фактически находится *(см. разд. 24.18)*. Например, при передаче из файла, расположенного в приложении `A`, в файл, расположенный в приложении `B`, приложение `A` фактически заимствует этот файл у приложения `B` и исполняет его как часть приложения `A`. Обратная сторона такой подмены — URL в браузере останется старым и не будет отражать истинного URL страницы, т. к. браузер клиента о подмене содержимого не знает.

24.6. Перенаправление на другую страницу при исключении

При использовании `Response.Redirect` есть одна тонкость, которую следует помнить при написании защищенного кода. Вызов `Redirect` внутри блока `try—catch` приведет к вызову исключения:

```
try
{
   // Некоторый код.
   // Обработка ошибки
   if (error)
    Response.Redirect("error.aspx");
   else
    Response.Redirect("nextpage.aspx");
  }
catch
{
    // Будет вызываться в любом случае
    AddLineToLog("Error");
}
```

В случае отсутствия "реального" исключения, будет вызвано исключение `Thread aborted`. Вызов `Redirect` следует выносить из блока `try-catch` или использовать метод `Transfer` или вызов `Redirect(url, true)`.

24.7. Использование cookies

Использование cookies показано в листинге 24.1.

Листинг 24.1. Использование cookies

```
private void ButtonRead_Click(object sender, System.EventArgs e)
{
  // Получаем объект cookie
  HttpCookie cookie = Request.Cookies["test_cookie"];
  // Если cookie существует, читаем запись test_text
  if (cookie!=null)
  TextBox1.Text = cookie["test_text"];
}
private void ButtonWrite_Click(object sender, System.EventArgs e)
{
  // Пробуем получить объект cookie
  HttpCookie cookie = Request.Cookies["test_cookie"];
  // Если такого объекта еще нет, то создаем
  if (cookie==null)
  cookie = new HttpCookie("test_cookie");
  // Записываем test_text
  cookie["test_text"] = TextBox1.Text;
  // Время жизни cookie - один год
  cookie.Expires = DateTime.Now.AddYears(1);
  // Сохраняем cookie
  Response.Cookies.Add(cookie);
}
```

24.8. Установка фокуса на элемент управления

Установка фокуса на элемент управления производится с помощью кода, приведенного в листинге 24.2.

Листинг 24.2. Установка фокуса на элемент управления

```
private void SetFocus(string ControlName)
 {
  // Добавляем функцию установки фокуса
  System.Text.StringBuilder sb = new System.Text.StringBuilder("");
  sb.Append("<script language=javascript>");
  sb.Append("function setFocus(ctl) {");
  sb.Append(" if (document.all[ctl] != null)");
  sb.Append(" {document.all[ctl].focus();}");
  sb.Append("}");
  // Добавляем вызов функции установки фокуса
  sb.Append("setFocus('");
```

```
  sb.Append(ControlName);
  sb.Append("');<");
  sb.Append("/");
  sb.Append("script>");
  // Регистрируем клиентский скрипт
  if (!Page.IsStartupScriptRegistered("InputFocusHandler"))
  Page.RegisterStartupScript("InputFocusHandler", sb.ToString());
}
private void Page_Load(object sender, System.EventArgs e)
{
  // Вызываем установку фокуса на TextBox1
  SetFocus(TextBox1.ClientID);
}
```

24.9. Просмотр исходного кода страницы

Для просмотра исходного кода страницы нужно в браузере набрать URL:

```
javascript:'<XMP>' + window.document.body.outerHTML + '</XMP>'
```

24.10. Открытие страницы по кнопке в новом окне

К кнопке добавляется обработчик `onclick`:

```
PrintLinkButton.Attributes["OnClick"] =
           "wnd('reportspr.aspx'); return false;";
```

В коде страницы или в присоединенном JS-файле должен быть описан соответствующий метод:

```
function wnd(url) { window.open(url); return false; }
```

24.11. Обработка исключений на странице

Обработку исключений на странице можно реализовать следующим способом (автор Крег Андера (Craig Andera)):

1. Добавить наследование интерфейса `IHttpHandler`.
2. Реализовать метод `ProcessRequest`.

Пример реализации показан в листинге 24.3.

Листинг 24.3. Обработка исключений на странице

```
public class DefaultPage : Page, IHttpHandler
{
  public new void ProcessRequest(HttpContext ctxt)
  {
    try
    {
      base.ProcessRequest(ctxt);
    }
    catch (Exception e)
    {
      ctxt.Response.Redirect(string.Format(
               "ErrorHandling.aspx?msg={0}", e.Message));
    }
}
```

24.12. Глобальная обработка исключений ASP.NET

Для глобальной обработки исключений необходимо реализовать собственный модуль `IHttpHandler` и зарегистрировать его в файле web.config (листинг 24.4):

```
<httpModules>
  <add type="Example.ErrorHandlingModule, Example"
                                          name="ErrorHandler" />
</httpModules>
```

См. также разд. 5.10 и 22.1.

Листинг 24.4. Модуль глобальной обработки исключений

```
using System;
using System.Text;
using System.Web;
using System.Web.UI;

namespace Example
{
  public class ErrorHandlingModule : IHttpModule
  {
    private HttpApplication _application;
```

```
    // Инициализация
    void IHttpModule.Init(HttpApplication application)
    {
      _application = application;
      _application.Error += new EventHandler(ErrorHandler);
    }

    // Обработчик
    private void ErrorHandler(Object sender, EventArgs e)
    {
      bool Handled = true;

      // Получаем ошибку
      Exception ex = _application.Server.GetLastError();

      // Формируем сообщение
      StringBuilder msg = new StringBuilder();
      if (ex.InnerException != null)
      {
        msg.Append(ex.InnerException.Message);
        if (ex.InnerException.InnerException != null)
        {
          msg.AppendFormat(" {0}",
                          ex.InnerException.InnerException.Message);
          Handled  =  false;
        }
      }

      // Передаем сообщение на страницу отображения ошибки
      if (Handled)
      {
        // Чистим ошибки
        _application.Server.ClearError();
        msg = msg.Replace("\r\n", " ");
        _application.Context.Response.Redirect(
             string.Format("./ErrorHandling.aspx?msg={0}", msg));
      }
    }
    public void Dispose()
    {
    }
  }
}
```

24.13. Ссылка на файл для скачивания

Один из способов создания ссылки на файл для скачивания — запись файла на диск и формирование ссылки с помощью метода `Server.MapPath` *(см. разд. 24.1)*. Недостаток этого способа — необходимость иметь права для записи файлов на диск. Кроме того, это довольно медленный способ.

Второй вариант — создание объекта `IHttpHandler` для создания обработчика формирования файла в памяти, без записи на диск. Такой обработчик регистрируется в файле web.config в секции `httpHandlers`:

```
<httpHandlers>
  <add verb="*" path="DownloadTxt.aspx"
                type="Example.DownloadTxtHandler,Example" />
</httpHandlers>
```

Ссылка на файл формируется как обычная ссылка на страницу, указанную в поле `path` обработчика. При обращении к файлу приложение будет передавать управление обработчику, указанному в поле `type`. Код обработчика приведен в листинге 24.5.

Листинг 24.5. Обработчик ссылки на файл для скачивания

```
using System.Text;
using System.Web;

namespace Example
{
  public class DownloadTxtHandler: IHttpHandler
  {
    bool IHttpHandler.IsReusable
    {
      get
      {
        return true;
      }
    }

    void IHttpHandler.ProcessRequest(HttpContext context)
    {
      string source_str  = "<b>Test</b>";
      byte[] source_data = Encoding.UTF8.GetBytes(source_str);

      context.Response.ContentType = "text/plain";
      context.Response.AddHeader("Content-Disposition",
                          "attachment; filename='test.txt'");
```

```
      context.Response.OutputStream.Write(source_data, 0,
                              source_data.Length);
      context.Response.End();
    }
  }
}
```

24.14. Управление созданием обработчиков *IHttpHandler*

С помощью класса `IHttpHandlerFactory` можно управлять созданием обработчиков. Вместо прямого создания `IHttpHanler` создается обработчик `IHttpHandlerFactory`, который регистрируется в файле web.config вместо обычного обработчика (листинг 24.6).

```
<httpHandlers>
  <!--add verb="*"   path="DownloadTxt.aspx"
         type="Example.DownloadTxtHandler,Example" /-->
      <add verb="GET" path="DownloadTxt.aspx"
           type="Example.DownloadHandlerFactory, Example" />
</httpHandlers>
```

Далее, внутри этого класса можно управлять выбором обработчика в зависимости от запроса (`get`, `post`), пути и т. д.

Листинг 24.6. Управление обработчиками с помощью класса `IHttpHandlerFactory`

```
using System;
using System.Web;

namespace Example
{
  public class DownloadHandlerFactory : IHttpHandlerFactory
  {
    public IHttpHandler GetHandler(HttpContext context,
           string requestType, String url, String pathTranslated)
    {
      IHttpHandler handlerToReturn = new DownloadTxtHandler();

        // Можно создавать различные обработчики
        // для post, get и т. д.
```

```
//      switch (context.Request.RequestType.ToLower())
//      {
//        case "get":
//          handlerToReturn = ...;
//        case "post":
//          handlerToReturn = ...;
//      }

      return handlerToReturn;
    }

    public void ReleaseHandler(IHttpHandler handler)
    {
    }

    public bool IsReusable
    {
      get
      {
        // Для использования пула нужно вернуть true
        return false;
      }
    }

  }
}
```

24.15. Сложная привязка данных с помощью *DataBinder*

Если обычных возможностей метода `DataBinder.Eval` не хватает (например, надо выполнить сложное форматирование данных или их дополнительную обработку), можно "обернуть" вызов в дополнительный метод. Такой метод должен возвращать тип `string` и принимать параметры типа `object`.

В ASPX-коде:

```
<%# ExFormatter(DataBinder.Eval(Container.DataItem, "Number"))%>
```

В CS-коде:

```
public string ExFormatter(object Number)
{
  int N = Convert.ToInt32(Number);
  return (N+1).ToString();
}
```

24.16. Что плохого в использовании сессий?

Механизм сессий, несомненно, является одним из самых удобных механизмов хранения данных между запросами. Однако следует помнить и о его ограничениях:

- каждая сессия занимает примерно 10 Кбайт (не считая самих данных, сохраненных в сессии) и немного замедляет выполнение каждого запроса;
- объект `Session`, сохраняемый на сервере, не может распространяться на все серверы при использовании мультисерверного Web-сайта. При использовании сессий следует продумать механизм запросов пользователя так, чтобы пользователь всегда работал с одним сервером, поскольку на другом сервере его данных просто нет;
- объект `Application` также не охватывает все серверы.

Если приложение не использует сессию, то для повышения производительности приложения, следует отключить ее использование.

В файле web.config можно настроить место расположения данных сессии: на сервере, на клиенте или в базе данных *(см. разд. 24.17)*.

Для данных небольшого объема рекомендуется использовать другие способы хранения: cookies, переменные `QueryString` или скрытые поля.

Еще один способ передачи данных с одной страницы на другую описан в *разд. 24.18*.

24.17. Настройка хранения сессий

Параметр `sessionState` в файле web.config позволяет настроить хранение данных сессии.

```
<configuration>
   <system.web>
      <sessionState>
         <sessionState mode="Off|InProc|StateServer|SQLServer"
              cookieless="true|false"
              timeout="number of minutes"
              stateConnectionString="tcpip=server:port"
              sqlConnectionString="sql connection string"
              stateNetworkTimeout="number of seconds"/>
```

Атрибут `mode` является обязательным и может принимать следующие значения:

- `Off` — указывает, что состояние сессии не включено;
- `InProc` — указывает, что состояние сессии сохраняется локально;

- `StateServer` — указывает, что состояние сессии сохраняется на удаленном сервере;
- `SQLServer` — указывает, что состояние сессии сохраняется на SQL Server.

Дополнительные атрибуты:

- `cookieless` — указывает, будут ли для идентификации сессий клиента использоваться сессии без cookie;
- `timeout` — задает число минут бездействия сессии перед ее удалением. Значение по умолчанию — 20;
- `stateConnectionString` — задает имя и порт сервера, на котором будет удаленно сохраняться состояние сессии. Например, `tcpip=127.0.0.1:42424`. Этот атрибут является обязательным, если атрибут `mode` имеет значение `StateServer`;
- `sqlConnectionString` — задает строку подключения для SQL Server. Например, `data source=localhost;Integrated Security=SSPI;Initial Catalog=northwind`. Этот атрибут является обязательным, если атрибут `mode` имеет значение `SQLServer`;
- `stateNetworkTimeout` — при использовании режима `StateServer` для хранения состояния сессии этот атрибут задает число секунд неактивности подключения TCP/IP между Web-сервером и сервером хранения состояния перед удалением сессии. Значение по умолчанию — 10.

Для использования режима `StateServer` должна быть запущена служба хранения состояния ASP.NET, содержащая сведения о состоянии сессии. Эта служба устанавливается вместе с ASP.NET и по умолчанию располагается в \systemroot\Microsoft.NET\Framework\version\aspnet_state.exe.

Для использования режима `SQLServer` должна быть создана база данных ASPState с помощью скрипта InstallSqlState.sql.

24.18. Передача между страницами значений серверного элемента управления

Для передачи данных между страницами можно использовать метод `Transfer` *(см. разд. 24.5)*. Общая последовательность действий такова:

1. Задайте имя исходной страницы с помощью директивы `Page`.

   ```
   <%@ Page Language="C#" ClassName="MyClassName" %>
   ```

2. Для каждого значения класса, которое требуется передать, создайте свойство с методом доступа `get`. Метод доступа `get` должен возвращать передаваемое значение, например, текст в поле ввода.

```
<script runat="server">
  public string FirstName
  {
    get
    {
      // FirstNameTextBox – это имя элемента управления TextBox
      return FirstNameTextBox.Text;
    }
  }
</script>
```

3. Для передачи данных на другую страницу Web Forms вызовите метод `HttpServerUtility.Transfer`.

```
void SubmitButtonClick(object sender, EventArgs e)
{
  Server.Transfer("secondpage.aspx");
}
```

4. Добавьте в начало принимающей страницы директиву `@ Reference`, а значение атрибута `Page` задайте равным имени страницы, передающей данные.

```
<%@ Reference Page="firstpage.aspx" %>
```

5. В серверном сценарии объявите переменную для хранения экземпляра класса, определенного на отправляющей странице Web Forms.

```
<script runat="server">
  FirstPageClass fp;
</script>
```

6. Создайте пользовательский обработчик события `Page_Load`.

```
<script runat="server">
  void Page_Load()
  {
     if (!IsPostBack)
     {
        fp = (FirstPageClass)Context.Handler;
     }
  }
</script>
```

7. Полученную ссылку на форму можно использовать для получения данных.

```
Hello <%=fp.FirstName%>
```

24.19. Как перехватить загрузку и сохранение *ViewState*?

Для обработки загрузки и сохранения `ViewState` следует перекрыть методы `SavePageStateToPersistenceMedium` и `LoadPageStateFromPersistenceMedium` (листинг 24.7).

Листинг 24.7. Сохранение и восстановление `ViewState`

```
public class MyPageTemplate : Page
{
   protected override void SavePageStateToPersistenceMedium(
                                  object viewState)
   {
     // Объект для получения ViewState
     LosFormatter los = new LosFormatter();
     // Сериализация
     StringWriter writer = new StringWriter();
     los.Serialize(writer, viewState);
     // Получаем строку
     string strViewState = writer.ToString();
     // Здесь можно сохранить строку ViewState, например в БД
   }
   protected override object LoadPageStateFromPersistenceMedium()
   {
     // Получили строку ViewState, например из БД
     string strViewState = ... ... ...;
     // Получаем объект ViewState
     LosFormatter los = new LosFormatter();
     return los.Deserialize(viewStateString);
   }
}
```

24.20. Создание общей сессии между ASP- и ASP.NET-приложениями

Создание общей сессии между классическим ASP-приложением и приложением ASP.NET описано на сайте **www.codeproject.com/useritems/SessionManager.asp**.

24.21. Что такое AJAX?

AJAX — это технология асинхронных запросов на сервер, позволяющая обойтись без перерисовки всей страницы (т. е. без postback).

Для ASP 1.0 придется напрямую создавать ActiveX-компонент `Microsoft.XMLHTTP`.

Листинги 24.8—24.10 демонстрируют пример использования этого компонента. Форма Page1.aspx содержит кнопку `btnGetData` и метку `labelData`. Данные возвращаются страницей Data1.aspx.

Для получения сложного набора данных можно использовать XML-формат ответа, как показано в листингах 24.11 и 24.12 (страницы Page2, Data2).

Для автоматического обновления страницы используется код, приведенный в листинге 24.13.

В зависимости от браузера применяются разные способы создания основного компонента (листинг 24.14, автор Ян Сьютл (Ian Suttle)).

Листинг 24.8. Код страницы Page1.aspx

```
<%@ Page language="c#" Codebehind="Page1.aspx.cs"
   AutoEventWireup="false" Inherits=" AJAX_Test1.Page1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>TestButton</title>
    <meta name="GENERATOR" Content="Microsoft
                                    Visual Studio .NET 7.1">
    <meta name="CODE_LANGUAGE" Content="C#">
    <meta name="vs_defaultClientScript"  content="JavaScript">
    <meta name="vs_targetSchema"
          content="http://schemas.microsoft.com/intellisense/ie5">
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form id="Form1" method="post" runat="server">
      <asp:Button ID="btnGetData" runat="server" Text="Button" />
      <asp:Label ID="labelData" runat="server"
                                Text="???"></asp:Label>
    </form>
  </body>
</HTML>
```

Листинг 24.9. Код Page1.aspx.cs

```
using System;
using System.Collections;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace AJAX_Test1
{
  public class Page1 : System.Web.UI.Page
  {
    protected System.Web.UI.WebControls.Button btnGetData;
    protected System.Web.UI.WebControls.Label labelData;

    private void Page_Load(object sender, System.EventArgs e)
    {
      RegisterScripts();
      btnGetData.Attributes.Add("OnClick", "return GetAJAXData();");
    }

    private const string strGetAJAXData = @"
     <script language='javascript'>
      var xRequest;
      function GetAJAXData()
      {
        if (window.XMLHttpRequest)
        {
          xRequest= new XMLHttpRequest();
        }
        else
        {
          if (typeof ActiveXObject != 'undefined')
          {
            xRequest= new ActiveXObject('Microsoft.XMLHTTP');
          }
        }

        if (xRequest != null)
        {
          xRequest.onreadystatechange = ProcessResponse;
          xRequest.open('GET', '%DATA_PAGE%',aatrue);
          xRequest.send(null);aaaaaaaa
        }
```

```
    return false;
  }
  function ProcessResponse()
  {
    if(xRequest.readyState == 4)
    {
      if(xRequest.status == 200)
      {
     var retval = xRequest.responseText;
        if (document.getElementById('%LABEL_NAME%') != null)
          document.getElementById('%LABEL_NAME%').innerHTML =
                                                    retval;
      }
      else
      {
        alert('Ошибка получения данных!');
      }
    }
  }
  </script>
";

private void RegisterScripts()
{
  if (!Page.IsStartupScriptRegistered("GetAJAXData"))
  {
    string script = strGetAJAXData;
    script = script.Replace("%DATA_PAGE%", "Data1.aspx");
    script = script.Replace("%LABEL_NAME%", labelData.ClientID);
    Page.RegisterStartupScript("GetAJAXData", script);
  }
}

#region Web Form Designer generated code
override protected void OnInit(EventArgs e)
{
  InitializeComponent();
  base.OnInit(e);
}

private void InitializeComponent()
{
  this.Load += new System.EventHandler(this.Page_Load);
}
```

```
    #endregion
  }
}
```

Листинг 24.10. Код Data1.aspx.cs, возвращающий данные

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace AJAX_Test1
{
  public class Data1 : System.Web.UI.Page
  {
    private void Page_Load(object sender, System.EventArgs e)
    {
      // Не сохранять
      Response.Cache.SetNoStore();
      // Не сохранять в истории браузера
      Response.Cache.SetAllowResponseInBrowserHistory(false);
      // Не кэшировать на сервере
      Response.Cache.SetNoServerCaching();

      Response.Clear();
      Response.Write(string.Format("Текущее время: {0}",
                                   DateTime.Now));
      Response.End();
    }
    #region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
      InitializeComponent();
      base.OnInit(e);
    }
```

```
    private void InitializeComponent()
    {
      this.Load += new System.EventHandler(this.Page_Load);
    }
    #endregion
  }
}
```

Листинг 24.11. Код Page2.aspx.cs (обработка данных в XML-формате)

```
... ... ... ... ... ... ... ... ... ...
function ProcessResponse()
{
  if(xRequest.readyState == 4)
  {
    if(xRequest.status == 200)
    {
      var retval = xRequest.responseText;
      if (xRequest.responseXML.loadXML(xRequest.responseText))
      {
        var value =
            xRequest.responseXML.getElementsByTagName
                                ('value')[0].firstChild.data;
       if (document.getElementById('%LABEL_NAME%') != null)
         document.getElementById('%LABEL_NAME%').innerHTML = value;
      }
      else
      {
        alert('Ошибка в XML-ответе!');
      }
    }
    else
    {
      alert('Ошибка получения данных!');
    }
  }
}
... ... ... ... ... ... ... ... ... ...
```

Листинг 24.12. Код Data2.aspx.cs (возврат данных в XML-формате)

```
... ... ... ... ... ... ... ... ... ...
private void Page_Load(object sender, System.EventArgs e)
```

```
{
... ... ... ... ... ... ... ... ... ...
  Response.Clear();

  // Сохраняем в виде XML
  string strXmlFormat = @"<response><value>{0}</value></response>";
  string strXmlValue =
                  string.Format(strXmlFormat, DateTime.Now.Second);
  XmlDocument result = new XmlDocument();
  result.LoadXml(strXmlValue);
  result.Save(Response.OutputStream);
  Response.End();
}
... ... ... ... ... ... ... ... ... ...
```

Листинг 24.13. Автоматическое обновление страницы

```
private const string strGetAJAXData = @"
  <script language='javascript'>
    var xRequest;
    var oInterval='';
    function GetAJAXData()
    {
      if (oInterval == '')
      {
         oInterval = window.setInterval('GetAJAXData()', 1000);
      }
      if (window.XMLHttpRequest)
... ... ... ... ... ... ... ... ... ...
```

Листинг 24.14. Создание основного компонента в зависимости от браузера

```
function GetXmlHttpObject(handler)
{
  var objXmlHttp = null; // для хранения ссылки

  if (is_ie)
  {
    // Разные версии IE
    var strObjName = (is_ie5) ?
                     'Microsoft.XMLHTTP' : 'Msxml2.XMLHTTP';
    // Пытаемся создать объект
    try
```

```
    {
      objXmlHttp = new ActiveXObject(strObjName);
      objXmlHttp.onreadystatechange = handler;
    }
    catch(e)
    {
      // Ошибка
      alert('Браузер IE, но объект создать не удалось.');
      return;
    }
  }
  else
    if (is_opera)
    {
      // В Opera могут быть проблемы
      alert('Браузер Opera. Страница может не работать.');
      return;
    }
    else
    {
      // Mozilla или Netscape, или Safari
      objXmlHttp = new XMLHttpRequest();
      objXmlHttp.onload  = handler;
      objXmlHttp.onerror = handler;
    }
  // Возвращаем объект
  return objXmlHttp;
}
```

24.22. Получение серверных данных без AJAX

Для получения серверных данных без postback можно воспользоваться методом, работающим с помощью IFRAME (автор Раджеш Толети (Rajesh Toleti)).

Листинг 24.15 содержит код страницы default.aspx, получающей данные со страницы Data.aspx (листинги 24.16 и 24.17).

Разумеется, этот код будет работать только в браузерах, поддерживающих IFRAME.

Листинг 24.15. Страница default.aspx

```
<%@ Page language="c#" Codebehind="default.aspx.cs"
              AutoEventWireup="false" Inherits="NoAJAX.WebForm1" %>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>WebForm1</title>
    <meta name="GENERATOR" Content="Microsoft
                                    Visual Studio .NET 7.1">
    <meta name="CODE_LANGUAGE" Content="C#">
    <meta name="vs_defaultClientScript" content="JavaScript">
    <meta name="vs_targetSchema"
            content="http://schemas.microsoft.com/intellisense/ie5">
    <script language="JavaScript">
      function callServer()
      {
        serverData.innerHTML = '<IFRAME src="Data.aspx?ID='+
          document.Form1.DropDown.value+
          '" width="200" height="50" frameborder="0"
                                    scrolling="no"></IFRAME>';
      }
    </script>

  </HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
      <table height="100" border="0">
        <tr>
          <td valign="middle">
            <select name="DropDown" onchange="callServer()">
              <option value="0" selected>Выбор...</option>
              <option value="1">Value1</option>
              <option value="2">Value2</option>
              <option value="3">Value3</option>
            </select>
          </td>
          <td valign="middle">
            <div id="serverData"></div>
          </td>
        </tr>
      </table>
    </form>
  </body>
</HTML>
```

Листинг 24.16. Код Data.aspx.cs (источник данных)

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace NoAJAX
{
  public class Data : System.Web.UI.Page
  {
    protected System.Web.UI.WebControls.Label DataLabel;

    private void Page_Load(object sender, System.EventArgs e)
    {
      string str = string.Empty;
      switch (Request.QueryString["ID"])
      {
        case "1": str="X1";  break;
        case "2": str="X2";  break;
        case "3": str="X3";  break;
      }

      DataLabel.Text = str;
    }

    #region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
      InitializeComponent();
      base.OnInit(e);
    }

    private void InitializeComponent()
    {
      this.Load += new System.EventHandler(this.Page_Load);
    }
```

```
    #endregion
  }
}
```

Листинг 24.17. Страница Data.aspx (источник данных)

```
<%@ Page language="c#" Codebehind="Data.aspx.cs"
                AutoEventWireup="false" Inherits="NoAJAX.Data" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
   <title>Data</title>
     <meta name="GENERATOR" Content="Microsoft
                 Visual Studio .NET 7.1">
     <meta name="CODE_LANGUAGE" Content="C#">
     <meta name="vs_defaultClientScript" content="JavaScript">
     <meta name="vs_targetSchema"
           content="http://schemas.microsoft.com/intellisense/ie5">
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form id="Form1" method="post" runat="server">
      <asp:label id="DataLabel" runat="server" />
    </form>
  </body>
</HTML>
```

24.23. Получение HTML-кода элемента

Следующий код возвращает строку, содержащую HTML-представление элемента:

```
public static string RenderControl(Control ctrl)
{
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  System.IO.StringWriter tw = new System.IO.StringWriter(sb);
  System.Web.UI.HtmlTextWriter hw =
                   new System.Web.UI.HtmlTextWriter(tw);
  ctrl.RenderControl(hw);
  return sb.ToString()
}
```

24.24. Печать страницы на принтер по умолчанию

Листинг 24.18 содержит код страницы, состоящей из двух кнопок — кнопка печати и кнопка параметров печати.

Листинг 24.18. Печать и параметры печати

```
<%@ Page language="c#" Codebehind="PrintTest.aspx.cs"
AutoEventWireup="false" Inherits="Example.PrintTest" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >

<html>
  <head>
    <title>PrintTest</title>
    <meta name="GENERATOR" Content="Microsoft Visual
                                              Studio .NET 7.1">
    <meta name="CODE_LANGUAGE" Content="C#">
    <meta name=vs_defaultClientScript content="JavaScript">
    <meta name=vs_targetSchema
          content="http://schemas.microsoft.com/intellisense/ie5">

    <object ID="wb" WIDTH=300 HEIGHT=151
        CLASSID="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2"
                                        VIEWASTEXT>
         <param name="ExtentX" value="7938">
         <param name="ExtentY" value="3986">
         <param name="ViewMode" value="0">
         <param name="Offline" value="0">
         <param name="Silent" value="0">
         <param name="RegisterAsBrowser" value="0">
         <param name="RegisterAsDropTarget" value="1">
         <param name="AutoArrange" value="0">
         <param name="NoClientEdge" value="0">
         <param name="AlignLeft" value="0">
         <param name="NoWebView" value="0">
         <param name="HideFilenames" value="0">
         <param name="SingleClick" value="0">
         <param name="SingleSelection" value="0">
         <param name="NoFolders" value="0">
         <param name="Transparent" value="0">
         <param name="ViewID"
                  value="{0057D0E0-3573-11CF-AE69-08002B2E1262}">
      </object>
```

```
      <script language="JavaScript">
        function PrintPage()
        {
          wb.ExecWB(6, 2, 2, 2);
        }

        function SetupPage()
        {
          wb.ExecWB(8, 0, 0, 0);
        }
      </script>

  </head>
  <body>
    <form id="Form1" method="post" runat="server">
          <input type="button" value="Print Page"
                 onclick="PrintPage()">
          <input type="button" value="Print Setup"
                 onclick="SetupPage()">
     </form>
  </body>
</html>
```

24.25. Проблемы несовместимости браузеров

Для создания HTML-кода для браузеров, несовместимых с Internet Explorer, ASP.NET использует класс `Html32TextWriter`, который создает элементы, согласно спецификации HTML 3.2. Для исправления этого ограничения следует добавить следующую строку в файл web.config:

```
<system.web>
  <browserCaps>
    tagwriter=System.Web.UI.HtmlTextWriter
  </browserCaps>
</system.web>
```

24.26. Быстрая генерация табличного представления Web-формы

По умолчанию Visual Studio генерирует расположение элементов с помощью стилей. Для того чтобы быстро перевести страницу на табличное представление, следует выполнить следующие шаги:

1. Открыть свойства Web-формы и установить параметр **Target Schema** в значение **Internet Explorer 3.02**, параметр **Page Layout** — в значение `GridLayout`.
2. Перейти на вкладку HTML-представления формы и удалить строку `ms_positioning="GridLayout"` из тега `body`.
3. Переключить **Target Schema** на значение по умолчанию **Internet Explorer 5.0**.

24.27. Почему *Page_Load* вызывается два раза?

Скорее всего, включено свойство страницы `AutoEventWireUp`.

24.28. Получение пути к странице

Для получения пути к странице можно воспользоваться методом `Request.Url.AbsoluteUri`. Нужно только не забыть удалить из него параметры, если они есть:

```
public string RequestUrl()
{
  string result = Request.Url.AbsoluteUri;
  if (Request.Url.Query.Length > 0)
    result = result.Replace(Request.Url.Query, string.Empty);
   return result;
 }
```

ГЛАВА 25

MS Office, Internet Explorer

25.1. Получение данных со страницы MS Excel

Для получения данных со страницы Excel можно использовать классы `DataSet` и `OleDbConnection` (листинг 25.1).

Листинг 25.1. Получение данных со страницы MS Excel

```
using System;
using System.Data;
using System.Data.OleDb;

namespace ExcelDataTest
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Имя файла
      string filename = @"book1.xls";

      // Строка подключения
      string ConnectionString= String.Format(
        "Provider=Microsoft.Jet.OLEDB.4.0;Extended
                    Properties=\"Excel 8.0;HDR=No\";
                    Data Source={0}", filename);
      // Открываем соединение
      DataSet   ds=new DataSet("EXCEL");
```

```
      OleDbConnection cn=new OleDbConnection(ConnectionString);
      cn.Open();

      // Получаем список листов в файле
      DataTable schemaTable =
        cn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,
            new object[] {null, null, null, "TABLE"});

      // Показать список листов в файле
      for (int i=0; i< schemaTable.Rows.Count; i++)
      {
        // Имена листов
        Console.WriteLine(schemaTable.Rows[i].ItemArray[2]);

        // Дата модификации
        Console.WriteLine(schemaTable.Rows[i].ItemArray[7]);
      }

      // Берем название первого листа
      string sheet1 = (string) schemaTable.Rows[0].ItemArray[2];
      // Выбираем все данные с листа
      string select = String.Format("SELECT * FROM [{0}]", sheet1);
      OleDbDataAdapter ad = new OleDbDataAdapter(select, cn);
      ad.Fill(ds);
      DataTable tb=ds.Tables[0];

      // Показать данные с листа
      foreach (DataRow row in tb.Rows)
      {
        foreach(object col in row.ItemArray)
        {
          Console.Write(col+"\t");
        }
        Console.WriteLine();
      }
    }
  }
}
```

25.2. Доступ к данным MS Excel

Листинг 25.2 содержит пример доступа к данным MS Excel. Для работы этого кода требуется подключение библиотеки **Microsoft Excel *xx* Object Library**.

Листинг 25.2. Запись значения ячейки Excel

```
using Excel;
using System;
using System.IO;
using System.Reflection;

namespace ExcelReader
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      ApplicationClass application = null;
      string     filename;
      Workbook   workbook;
      Worksheet  worksheet;
      object     objsheet, objrange;
      Range      range;

      // Должно быть полное имя
      filename = Path.Combine(
            System.AppDomain.CurrentDomain.BaseDirectory,
            @"test.xls");

      try
      {
        application = new ApplicationClass();
        application.Visible=false;
        application.DisplayAlerts=false;

        workbook = application.Workbooks.Open(
          filename,
          Missing.Value,
          Missing.Value,
          Missing.Value,
          Missing.Value,
          Missing.Value,
          Missing.Value,
          Missing.Value,
          Missing.Value,
          Missing.Value,
```

```
          Missing.Value,
          Missing.Value,
          Missing.Value
          );

        objsheet = workbook.ActiveSheet;
        if (objsheet != null)
        {
          worksheet = (Worksheet) objsheet;
          objrange = worksheet.Cells[1, 1];
          if (objrange != null)
          {
            range = (Range) objrange;
            range.Font.Name="Tahoma";
            range.Font.Size=8;
            range.Font.Bold=false;
            range.Value = "новое значение";
            range = null;
          }
          objrange = null;
          worksheet = null;
        }
        objsheet = null;

        workbook.Save();
        workbook = null;
      }
      catch(Exception ex)
      {
        Console.WriteLine(ex.ToString());
      }
      finally
      {
        if (application != null)
        {
          application.Quit();
        }
        application = null;
      }
    }
  }
}
```

25.3. Работа с данными в буфере обмена в Excel-формате CSV

Пример из листинга 25.3 показывает копирование и получение данных из буфера обмена в формате Excel.

Листинг 25.3. Получение данных из буфера обмена в формате Excel

```
System.Windows.Forms.IDataObject o =
         System.Windows.Forms.Clipboard.GetDataObject();
if (o.GetDataPresent(System.Windows.Forms.
                          DataFormats.CommaSeparatedValue))
{
  System.IO.StreamReader sr =
            new System.IO.StreamReader((System.IO.Stream)
            o.GetData(DataFormats.CommaSeparatedValue));
  string s = sr.ReadToEnd();
  sr.Close();
  label1.Text= s;
 }
// Копирование данных в буфер обмена
// В зависимости от "list separator".
// Может быть разделение запятой
String csv = "1;2;3" + Environment.NewLine + "6;8;3";
byte[] blob = System.Text.Encoding.UTF8.GetBytes(csv);
System.IO.MemoryStream s = new System.IO.MemoryStream(blob);
DataObject data = new DataObject();
data.SetData(DataFormats.CommaSeparatedValue, s);
Clipboard.SetDataObject(data, true);
```

25.4. Как обойти проблемы с использованием объектов MS Office?

Обходной маневр такой: надо настройку региональных стандартов текущего потока переключить на английскую. А после использования объектов MS Office — не забыть вернуть на место.

```
System.Globalization.CultureInfo oldCulture =
        System.Threading.Thread.CurrentThread.CurrentCulture;
Thread.CurrentThread.CurrentCulture =
        new System.Globalization.CultureInfo( "en-US" );
try
```

```
{
  // Тут вызываем методы MS Office
}
finally
{
  System.Threading.Thread.CurrentThread.CurrentCulture = oldCulture;
}
```

25.5. Как напечатать документ MS Office?

Для печати документа MS Office проще всего использовать класс `Process`.

```
System.Diagnostics.Process pr = new System.Diagnostics.Process();
pr.StartInfo.Verb = "Print";
pr.StartInfo.FileName = "Sample.xls";
pr.Start();
```

25.6. Формирование документов MS Excel и MS Word из ASP.NET

Формирование документов Excel из ASP.NET выполняется очень просто: для этого необходимо указать соответствующий `ContentType` (листинг 25.4).

При необходимости можно указывать формулы в атрибутах тегов (листинг 25.5). В этом случае документ, открытый в браузере, будет содержать число, а открытый в Excel — формулу:

```
"<TDCOLSPAN=2 FORMULA=\"=A2+B2\">310</TD>"
```

Имя файла открываемого документа можно указать с помощью строки:

```
Response.AddHeader("Content-Disposition", "attachment;
                                          filename=Test.xls");
```

Аналогично можно формировать документы MS Word:

```
Response.ContentType = "application/msword";
```

Существуют два варианта открытия документа: в том же окне браузера или как файл. В первом случае нужно дописать строку:

```
Response.AddHeader("Content-Disposition", "inline;
                           filename=report.xls");
```

а во втором:

```
Response.AddHeader("Content-Disposition", "attachment;
                           filename=report.xls");
```

Для данных большого объема рекомендуется использовать `inline`-метод, а для того чтобы отчет открывался в новом окне, писать свой обработчик перехода на страницу отчета:

```
function wnd(url) { window.open(url); return false; }
```

Следует учитывать, что при формировании документа должно быть включено кэширование формирующей страницы.

Листинг 25.4. Формирование документа Excel

```
private void Page_Load(object sender, System.EventArgs e)
{
  // Формирование таблицы (можно читать из файла
  // или формировать программно)
  string text =
    "<HTML>" +
      "<HEAD>" +
        "<TITLE>Пример</TITLE>" +
      "</HEAD>"+
      "<BODY>"+
        "<TABLE BORDER=\"1\">"+
          "<TR>"+
            "<TD WIDTH=70>Title 1</TD>"+
            "<TD WIDTH=140>Title 2</TD>"+
          "</TR>"+
          "<TR>"+
            "<TD>155</TD>"+
            "<TD>200</TD>"+
          "</TR>"+
      "<BODY>"+
    "<HTML>";

  // Указываем ContentType
  Response.ContentType = "application/vnd.ms-excel";
  Response.Charset = "";
  Response.Write(text);
}
```

Листинг 25.5. Задание формул в описании Excel-документа на языке XML

```
string text =
  "<HTML>" +
   "<HEAD>" +
```

```
    "<TITLE>Пример</TITLE>" +
   "</HEAD>"+
   "<BODY>"+
    "<TABLE BORDER=\"1\">"+
     "<TR>"+
     "<TD WIDTH=70>Title 1</TD>"+
     "<TD WIDTH=140>Title 2</TD>"+
    "</TR>"+
    "<TR>"+
     "<TD>155</TD>"+
     "<TD>200</TD>"+
    "</TR>"+
    "<TR>"+
    "<TD COLSPAN=2>=A3+B3</TD>"+
    "</TR>"+
   "<BODY>"+
  "<HTML>";
```

25.7. Работа с MS Office через DDE

См. статью Гасанова Р. З. "Организация “горячего” обмена по DDE между Microsoft Excel и приложением .NET": **www.codenet.ru/progr/cpp/dotnet/Hot-DDE**.

25.8. Работа с календарем MS Outlook

Листинг 25.6 демонстрирует работу с календарем MS Outlook. Для работы кода требуется подключение библиотеки **Microsoft Outlook *xx* Object Library**.

Листинг 25.6. Работа с календарем MS Outlook

```
using System;
using System.Reflection;
using Microsoft.Office.Interop.Outlook;

namespace OutlookCalendar
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
```

```
    {
      ApplicationClass application = new ApplicationClass();

      NameSpace nspace = application.GetNamespace("MAPI");

      nspace.Logon(Missing.Value, Missing.Value, false, false);


      object objappointment =
               application.CreateItem(OlItemType.olAppointmentItem);

      if (objappointment != null)
      {
        AppointmentItem appointment =
                                 (AppointmentItem) objappointment;

        appointment.Importance = OlImportance.olImportanceHigh;
        appointment.Subject = "watch tv";
        appointment.Body = "Не забудь посмотреть телевизор!";
        appointment.Start = DateTime.Parse("1-1-2006 11:00 am");
        appointment.End = DateTime.Parse("1-1-2006 1:00 pm");
        appointment.Location = "Диван";
        appointment.ReminderSet = true;
        appointment.ReminderMinutesBeforeStart = 60;
        appointment.AllDayEvent = false;
        appointment.Save();

        appointment = null;
      }

      nspace.Logoff();
      nspace = null;

      application.Quit();
      application = null;
    }
  }
}
```

25.9. Использование Word Spell Checker для проверки орфографии

Листинг 25.7 демонстрирует использование Word Spell Checker для проверки орфографии. Для работы этого кода требуется подключение библиотеки

Microsoft Word *xx* Object Library (модуль Interop.Word.dll). Второй вариант (листинг 25.8) отображает стандартный диалог выбора слов для замены из словаря.

Листинг 25.7. Проверка орфографии средствами MS Word

```
using System;
using System.Collections;
using System.Reflection;
using Microsoft.Office.Interop.Word;

namespace WordSpellChecker
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Текст для проверки
      string sentence = "Проверка орфографии слуво с ошибкай.";

      // Делим текст на слова
      string[] words = sentence.Split();

      // Класс для проверки слов
      SpellChecker checker = new SpellChecker();

      // Проверяем каждое слово
      foreach (string word in words)
      {
        // Проверка
        string[] suggestions = checker.Suggest(word);

        // Правильно написано
        if (suggestions == null)
        {
          Console.WriteLine("\"{0}\" написано верно", word);
        }
        else
        {
          // Ошибка - предлагаем варианты
          Console.WriteLine("\"{0}\" написано не верно! Варианты:",
                                                          word);
```

```
        foreach (string suggestion in suggestions)
          Console.WriteLine("\t" + suggestion);
      }
      Console.WriteLine();
    }
    checker = null;
  }
}

// Класс для проверки слов
class SpellChecker
{
  private ApplicationClass application;

  public SpellChecker()
  {
    object template = "normal.dot";
    object newtemplate = false;
    object doctype = WdNewDocumentType.wdNewBlankDocument;
    object visible = false;

    // Объект MS word
    this.application = new ApplicationClass();
    this.application.DisplayAlerts = WdAlertLevel.wdAlertsNone;
    this.application.Visible = false;
    this.application.Options.SuggestSpellingCorrections = true;

    // Создаем новый документ
    Document document = this.application.Documents.Add(
         ref template, ref newtemplate, ref doctype, ref visible);
    document.Activate();
  }

  ~SpellChecker()
  {
    // Завершить работу word, ничего не сохраняя
    object savenochanges = WdSaveOptions.wdDoNotSaveChanges;
    object nothing = Missing.Value;

    if (this.application != null)
      this.application.Quit(
              ref savenochanges, ref nothing, ref nothing);
    this.application = null;
  }
```

```
    /// <summary>
    /// Проверка слова
    /// </summary>
    public string[] Suggest(string word)
    {
      object nothing = Missing.Value;

      // Проверяем
      bool spelledright = this.application.CheckSpelling(word,
            ref nothing, ref nothing, ref nothing, ref nothing,
            ref nothing, ref nothing, ref nothing, ref nothing,
            ref nothing, ref nothing, ref nothing, ref nothing);

      if (spelledright)
        return null;

      // Получаем список слов для замены
      SpellingSuggestions suggestions =
         this.application.GetSpellingSuggestions(word,
             ref nothing, ref nothing, ref nothing, ref nothing,
             ref nothing, ref nothing, ref nothing, ref nothing,
             ref nothing, ref nothing, ref nothing, ref nothing,
             ref nothing);

      // Сохраняем список слов
      ArrayList words = new ArrayList();
      foreach (SpellingSuggestion suggestion in suggestions)
        words.Add(suggestion.Name);

      suggestions = null;

      // Возвращаем массив слов
      return (string[]) words.ToArray(typeof (string));
    }
  }
}
```

Листинг 25.8. Отображение стандартного диалога Word для корректировки слов

```
using System;
using System.ComponentModel;
using System.Reflection;
```

```
using System.Windows.Forms;
using Microsoft.Office.Interop.Word;
using Application = Microsoft.Office.Interop.Word.Application;

namespace SpellCheckDemo
{
  public class Form1 : Form
  {
    private TextBox textBox1;
    private Button button1;
    private Label label1;
    private Container components = null;
... ... ... ... ... ... ... ... ... ... ... ...
    [STAThread]
    static void Main()
    {
      System.Windows.Forms.Application.Run(new Form1());
    }

    private void button1_Click(object sender, EventArgs e)
    {
      string input = textBox1.Text;
      label1.Text = fSpellCheck(ref input);
      textBox1.Text = input;
    }

    public string fSpellCheck(ref string text)
    {
      string result = string.Empty;

      int iErrorCount = 0;
      Application  app = new Application();
      if (text.Length > 0)
      {
        app.Visible=false;

        object template=Missing.Value;
        object newTemplate=Missing.Value;
        object documentType=Missing.Value;
        object visible=true;
        object optional = Missing.Value;

        _Document doc = app.Documents.Add(ref template,
                  ref newTemplate, ref documentType, ref visible);
```

```
        doc.Words.First.InsertBefore (text);
        ProofreadingErrors we =  doc.SpellingErrors;
        iErrorCount = we.Count;

        doc.CheckSpelling( ref optional, ref optional, ref optional,
              ref optional, ref optional, ref optional,
              ref optional, ref optional, ref optional,
              ref optional, ref optional, ref optional);
        if (iErrorCount == 0)
          result = "Нет ошибок";
        else
          result = string.Format("Исправлено {0} ошибок.",
                                          iErrorCount);
        object first=0;
        object last=doc.Characters.Count -1;

        text = doc.Range(ref first, ref last).Text;
      }
      object saveChanges = false;
      object originalFormat = Missing.Value;
      object routeDocument = Missing.Value;
      app.Quit(ref saveChanges, ref originalFormat,
                                          ref routeDocument);
      return result;
    }
  }
}
```

25.10. Журнал истории Internet Explorer

Листинг 25.9 содержит код, который выводит все адреса из истории браузера (автор Красин (Krasin)).

Листинг 25.9. Получение адресов из истории браузера

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

namespace IEHistory
{
  class Class1
  {
    struct STATURL
```

```
{
  public static uint SIZEOF_STATURL =
              (uint)Marshal.SizeOf(typeof(STATURL));
  public uint cbSize;
  [MarshalAs(UnmanagedType.LPWStr)]
  public string pwcsUrl;
  [MarshalAs(UnmanagedType.LPWStr)]
  public string pwcsTitle;
  public FILETIME ftLastVisited, ftLastUpdated, ftExpires;
  public uint dwFlags;
}

[ComImport, Guid("3C374A42-BAE4-11CF-BF7D-00AA006946EE"),
  InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
  interface IEnumSTATURL
{
  [PreserveSig]
  uint Next(uint celt, out STATURL rgelt,
                       out uint pceltFetched);
  void Skip(uint celt);
  void Reset();
  void Clone(out IEnumSTATURL ppenum);
  void SetFilter(
    [MarshalAs(UnmanagedType.LPWStr)] string poszFilter,
    uint dwFlags);
}

[ComImport, Guid("AFA0DC11-C313-11d0-831A-00C04FD5AE38"),
  InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
  interface IUrlHistoryStg2
{
  void AddUrl(
    [MarshalAs(UnmanagedType.LPWStr)] string pocsUrl,
    [MarshalAs(UnmanagedType.LPWStr)] string pocsTitle,
    uint dwFlags);

  void DeleteUrl(
    [MarshalAs(UnmanagedType.LPWStr)] string pocsUrl,
    uint dwFlags);

  void QueryUrl(
    [MarshalAs(UnmanagedType.LPWStr)] string pocsUrl,
    uint dwFlags,
    ref STATURL lpSTATURL);
```

```
      void BindToObject(
        [MarshalAs(UnmanagedType.LPWStr)] string pocsUrl,
        ref Guid riid,
        [MarshalAs(UnmanagedType.IUnknown)] out object ppvOut);

      IEnumSTATURL EnumUrls();

      void AddUrlAndNotify(
        [MarshalAs(UnmanagedType.LPWStr)] string pocsUrl,
        [MarshalAs(UnmanagedType.LPWStr)] string pocsTitle,
        uint dwFlags,
        [MarshalAs(UnmanagedType.Bool)] bool fWriteHistory,
        [MarshalAs(UnmanagedType.IUnknown)] object
        poctNotify,
        [MarshalAs(UnmanagedType.IUnknown)] object punkISFolder);

      void ClearHistory();
    }

    [ComImport, Guid("3C374A40-BAE4-11CF-BF7D-00AA006946EE")]
    class UrlHistory { }

    [STAThread]
    static void Main(string[] args)
    {
      IUrlHistoryStg2 uhs2 = (IUrlHistoryStg2)new UrlHistory();
      IEnumSTATURL estaturl = uhs2.EnumUrls();
      STATURL staturl;
      uint fetched;

      while (0 == estaturl.Next(1, out staturl, out fetched))
        Console.WriteLine(staturl.pwcsUrl);
    }
  }
}
```

ГЛАВА 26

WMI

26.1. Что такое WMI?

WMI (Windows Management Instrumentation) — это инструментарий управления Windows. Сайт **microsoft.ru** дает следующее определение: "это масштабируемая инфраструктура управления, использующая один целостный расширяемый объектно-ориентированный и основанный на стандартах интерфейс".

Классы WMI находятся в пространстве имен `System.Management`, модуль `System.Management.dll`. Следует учитывать, что WMI является строго Windows-функциональностью, т. е. платформозависимой.

Пространство имен WMI содержит следующую иерархию классов:

```
Namespace System.Management
       → EventQuery  → WqlEventQuery
       → ObjectQuery → WqlObjectQuery
```

26.2. Работа с WMI на удаленной машине

Листинг 26.1 показывает работу с функциями WMI на удаленной машине.

Листинг 26.1. Работа с функциями WMI на удаленной машине

```
using System;
using System.Management;

namespace WMITest
{
  static void Main(string[] args)
```

```
  {
    ConnectionOptions options = new ConnectionOptions();
    options.Username = @"domain\username";
    options.Password = "password";
    ManagementScope scope =
       new ManagementScope(@"\\machine_name\root\cimv2", options);
    scope.Connect();

    try
    {
      // Работа с функциями WMI
    }
    catch (Exception e)
    {
      // Обработка ошибок
    }
  }
}
```

26.3. Получение свойств видеоконтроллера и частоты обновления экрана

Использование запроса `Win32_VideoController` позволяет получить свойства видеоконтроллера и частоту обновления экрана (листинг 26.2).

Листинг 26.2. Получение свойств видеоконтроллера и частоты обновления экрана

```
using System;
using System.Management;

namespace Win32_VideoController
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      ManagementScope sc=
                    new ManagementScope(@"\\.\root\cimv2", null);
      ManagementPath ph=
                    new ManagementPath(@"Win32_VideoController");
```

```
      ManagementClass mc=
                    new ManagementClass(sc, ph, null);

      foreach(ManagementObject ss in mc.GetInstances())
      {
        Console.WriteLine("Name: {0}", ss.GetPropertyValue("Name"));
        Console.WriteLine(" Processor {0}",
                      ss.GetPropertyValue("VideoProcessor"));
        Console.WriteLine(" VideoRAM {0}",
                      ss.GetPropertyValue("AdapterRAM"));
        Console.WriteLine(" Resolution {0}",
                      ss.GetPropertyValue("VideoModeDescription"));
        Console.WriteLine(" Refresh {0}",
                      ss.GetPropertyValue("CurrentRefreshRate"));
      }
    }
  }
}
```

26.4. Получение информации о компьютере

Запрос `Win32_ComputerSystem` позволяет получить информацию о компьютере (листинг 26.3).

Листинг 26.3. Получение информации о компьютере

```
using System;
using System.Management;

namespace Win32_ComputerSystem
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query =
           new WqlObjectQuery("SELECT * FROM Win32_ComputerSystem");
      ManagementObjectSearcher find =
           new ManagementObjectSearcher(query);
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("Computer belongs to domain " +
                    mo["Domain"]);
```

```
        Console.WriteLine("Computer manufacturer."+
                    mo["Manufacturer"]);
        Console.WriteLine("Model name given by manufacturer " +
                    mo["Model"]);
      }
    }
  }
}
```

26.5. Получение информации о производителе

Запрос `Win32_ComputerSystemProduct` позволяет получить информацию о производителе (листинг 26.4).

Листинг 26.4. Получение информации о производителе

```
using System;
using System.Management;

namespace Win32_ComputerSystemProduct
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query = new WqlObjectQuery(
                "SELECT * FROM Win32_ComputerSystemProduct");
      ManagementObjectSearcher find =
                 new ManagementObjectSearcher(query);
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("Description." + mo["Description"]);
        Console.WriteLine("Identifying number (usually
                    serial number)." + mo["IdentifyingNumber"]);
        Console.WriteLine("Commonly used product name." +
                    mo["Name"]);
        Console.WriteLine("Universally Unique Identifier of
                    product." + mo["UUID"]);
        Console.WriteLine("Vendor of product." + mo["Vendor"]);
      }
    }
  }
}
```

26.6. Получение информации об операционной системе

Листинг 26.5 демонстрирует получение информации об операционной системе.

Листинг 26.5. Получение информации об операционной системе

```
using System;
using System.Management;

namespace Win32_OperatingSystem
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query =
         new WqlObjectQuery("SELECT * FROM Win32_OperatingSystem");
      ManagementObjectSearcher find =
         new ManagementObjectSearcher(query);
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("Boot device name " + mo["BootDevice"]);
        Console.WriteLine("Build number" + mo["BuildNumber"]);
        Console.WriteLine("Caption" + mo["Caption"]);
        Console.WriteLine("Code page used by OS" + mo["CodeSet"]);
        Console.WriteLine("Country code" + mo["CountryCode"]);
        Console.WriteLine("Latest service pack installed" +
                          mo["CSDVersion"]);
        Console.WriteLine("Computer system name" + mo["CSName"]);
        Console.WriteLine("Time zone (minute offset from GMT" +
                          mo["CurrentTimeZone"]);
        Console.WriteLine("OS is debug build" + mo["Debug"]);
        Console.WriteLine("OS is distributed across several nodes."+
                          mo["Distributed"]);
        Console.WriteLine("Encryption level of transactions" +
                          mo["EncryptionLevel"] + " bits");
        Console.WriteLine(">>Priority increase for foreground app" +
                          GetForeground(mo));
        Console.WriteLine("Available physical memory" +
                          mo["FreePhysicalMemory"] + " kilobytes");
```

```
Console.WriteLine("Available virtual memory" +
                  mo["FreeVirtualMemory"] + " kilobytes");
Console.WriteLine("Free paging-space withou swapping." +
                  mo["FreeSpaceInPagingFiles"]);
Console.WriteLine("Installation date " +
                  ManagementDateTimeConverter.ToDateTime(
                        mo["InstallDate"].ToString()));
Console.WriteLine("What type of memory optimization......" +
          (Convert.ToInt16(mo["LargeSystemCache"]) == 0 ?
          "for applications" : "for system performance"));
Console.WriteLine("Time from last boot " +
            mo["LastBootUpTime"]);
Console.WriteLine("Local date and time " +
            ManagementDateTimeConverter.ToDateTime(
            mo["LocalDateTime"].ToString()));
Console.WriteLine("Language identifier (LANGID) " +
            mo["Locale"]);
Console.WriteLine("Local date and time. " +
            ManagementDateTimeConverter.ToDateTime(
            mo["LocalDateTime"].ToString()));
Console.WriteLine("Max# of processes supported by OS" +
            mo["MaxNumberOfProcesses"]);
Console.WriteLine("Max memory available for process." +
            mo["MaxProcessMemorySize"] + " kilobytes");
Console.WriteLine("Current number of processes" +
            mo["NumberOfProcesses"]);
Console.WriteLine("Currently stored user sessions." +
            mo["NumberOfUsers"]);
Console.WriteLine("OS language version" + mo["OSLanguage"]);
Console.WriteLine("OS product suite version" + GetSuite(mo));
Console.WriteLine("OS type" + GetOSType(mo));
Console.WriteLine("Number of Windows Plus! " +
            mo["PlusProductID"]);
Console.WriteLine("Version of Windows Plus! " +
          mo["PlusVersionNumber"]);
Console.WriteLine("Type of installed OS. " +
            GetProductType(mo));
Console.WriteLine("Registered user of OS. " +
            mo["RegisteredUser"]);
Console.WriteLine("Serial number of product. " +
            mo["SerialNumber"]);
Console.WriteLine("Serial number of product. " +
            mo["SerialNumber"]);
```

```
    Console.WriteLine("ServicePack major version. " +
                mo["ServicePackMajorVersion"]);
    Console.WriteLine("ServicePack minor version. " +
                mo["ServicePackMinorVersion"]);
    Console.WriteLine("Total number to store in paging files" +
                mo["SizeStoredInPagingFiles"] + " kilobytes");
    Console.WriteLine("Status. " + mo["Status"]);
    Console.WriteLine("ServicePack minor version. " +
                mo["ServicePackMinorVersion"]);
    Console.WriteLine("OS suite. " + GetOSSuite(mo));
    Console.WriteLine("Physical disk partition with OS. " +
                mo["SystemDevice"]);
    Console.WriteLine("System directory. " +
                mo["SystemDirectory"]);
    Console.WriteLine("Total virtual memory. " +
                mo["TotalVirtualMemorySize"] + " kilobytes");
    Console.WriteLine("Total physical memory. " +
                mo["TotalVisibleMemorySize"] + " kilobytes");
    Console.WriteLine("Version number of OS. " + mo["Version"]);
    Console.WriteLine("Windows directory. " +
                mo["WindowsDirectory"]);
  }
}

private static string GetForeground(ManagementObject mo)
{
  int i = Convert.ToInt16(mo["ForegroundApplicationBoost"]);
  switch (i)
  {
    case 0:  return "None";
    case 1:  return "Minimum";
    case 2:  return "Maximum (defualt value)";
  }
  return "Boost not defined.";
}

private static string GetSuite(ManagementObject mo)
{
  uint i = Convert.ToUInt32(mo["OSProductSuite"]);
  switch (i)
  {
    case 1:   return "Small Business";
    case 2:   return "Enterprise";
    case 4:   return "BackOffice";
```

```
      case 8:   return "Communication Server";
      case 16:  return "Terminal Server";
      case 32:  return "Small Business (Restricted)";
      case 64:  return "Embedded NT";
      case 128: return "Data Center";
    }
    return "OS suite not defined.";
  }

  // Тип операционной системы
  private static string GetOSType(ManagementObject mo)
  {
    uint i = Convert.ToUInt16(mo["OSType"]);
    switch (i)
    {
      case 16:  return "WIN95";
      case 17:  return "WIN98";
      case 18:  return "WINNT";
      case 19:  return "WINCE";
    }
    return "Other OS systems are not covered.";
  }

  private static string GetProductType(ManagementObject mo)
  {
    uint i = Convert.ToUInt32(mo["ProductType"]);
    switch (i)
    {
      case 1:  return "Work Station";
      case 2:  return "Domain Controller";
      case 3:  return "Server";
    }
    return "Product type not defined.";
  }

  private static string GetOSSuite(ManagementObject mo)
  {
    uint i = Convert.ToUInt32(mo["SuiteMask"]);
    string suite = "";
    if ((i & 1) == 1) suite += "Small Business";
    if ((i & 2) == 2)
    {
      if (suite.Length > 0) suite += ", "; suite += "Enterprise";
    }
```

```
if ((i & 4) == 4)
{
  if (suite.Length > 0) suite += ", "; suite += "Back Office";
}
if ((i & 8) == 8)
{
  if (suite.Length > 0)
     suite += ", ";
  suite += "Communications";
}
if ((i & 16) == 16)
{
  if (suite.Length > 0)
    suite += ", ";
  suite += "Terminal";
}
if ((i & 32) == 32)
{
  if (suite.Length > 0)
    suite += ", ";
  suite += "Small Business Restricted";
}
if ((i & 64) == 64)
{
  if (suite.Length > 0)
    suite += ", ";
  suite += "Embedded NT";
}
if ((i & 128) == 128)
{
  if (suite.Length > 0)
    suite += ", ";
  suite += "Data Center";
}
if ((i & 256) == 256)
{
  if (suite.Length > 0)
    suite += ", ";
  suite += "Single User";
}
if ((i & 512) == 512)
{
  if (suite.Length > 0)
```

```
          suite += ", ";
        suite += "Personal";
      }
      if ((i & 1024) == 1024)
      {
        if (suite.Length > 0)
          suite += ", ";
        suite += "Blade";
      }
      return suite;
    }
  }
}
```

26.7. Выполнение logoff, shutdown, reboot

Листинг 26.6 показывает выполнение операций logoff, shutdown, reboot и т. д.

Листинг 26.6. Выполнение logoff, shutdown, reboot и т. д.

```
using System;
using System.Management;

namespace Win32_OperatingSystem_1
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      object[] FLAG_LOGOFF        =  {0};
      object[] FLAG_SHUTDOWN      =  {1};
      object[] FLAG_REBOOT        =  {2};
      object[] FLAG_FORCELOGOFF   =  {4};
      object[] FLAG_FORCESHUTDOWN =  {5};
      object[] FLAG_FORCEREBOOT   =  {6};
      object[] FLAG_POWEROFF      =  {8};
      object[] FLAG_FORCEPOWEROFF = {12};

      SelectQuery query = new SelectQuery("Win32_OperatingSystem");
      ManagementObjectSearcher find =
                              new ManagementObjectSearcher(query);
```

```
      try
      {
        // Выполняемая команда
        // указывается в командной строке приложения
        int mode = Convert.ToInt32(args[0]);

        foreach (ManagementObject mo in find.Get())
        {
          if (mode == (int)FLAG_LOGOFF[0])
            mo.InvokeMethod("Win32Shutdown", FLAG_LOGOFF);
          else if (mode == (int)FLAG_SHUTDOWN[0])
            mo.InvokeMethod("Win32Shutdown", FLAG_SHUTDOWN);
          else if (mode == (int)FLAG_REBOOT[0])
            mo.InvokeMethod("Win32Shutdown", FLAG_REBOOT);
          else if (mode == (int)FLAG_FORCELOGOFF[0])
            mo.InvokeMethod("Win32Shutdown", FLAG_FORCELOGOFF);
          else if (mode == (int)FLAG_FORCESHUTDOWN[0])
            mo.InvokeMethod("Win32Shutdown", FLAG_FORCESHUTDOWN);
          else if (mode == (int)FLAG_FORCEREBOOT[0])
            mo.InvokeMethod("Win32Shutdown", FLAG_FORCEREBOOT);
          else if (mode == (int)FLAG_POWEROFF[0])
            mo.InvokeMethod("Win32Shutdown", FLAG_POWEROFF);
          else if (mode == (int)FLAG_FORCEPOWEROFF[0])
            mo.InvokeMethod("Win32Shutdown", FLAG_FORCEPOWEROFF);
          else
            Console.WriteLine("Неизвестный режим.");
        }
      }
      catch (Exception e)
      {
        Console.WriteLine(e.Message);
      }
    }
  }
}
```

26.8. Получение информации о рабочем столе

Запрос `Win32_Desktop` используется для получения информации о рабочем столе (листинг 26.7).

Листинг 26.7. Получение информации о рабочем столе

```
using System;
using System.Management;
```

```
namespace Win32_Desktop
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Получить настройки рабочего стола
      WqlObjectQuery query = new WqlObjectQuery(
           "SELECT * FROM Win32_Desktop WHERE Name ='.Default'");
      ManagementObjectSearcher find =
              new ManagementObjectSearcher(query);
      foreach (ManagementObject mo in find.Get())
      {
        // Значения могут быть изменены
        // в реестре "HKEY_CURRENT_USER\Control Panel\Desktop"
        Console.WriteLine("Width of window borders." +
                       mo["BorderWidth"]);
        Console.WriteLine("ALT+TAB task switching allowed." +
                       mo["CoolSwitch"]);
        // Значения в миллисекундах
        Console.WriteLine("Lenght of time between
              cursor blincks. " + mo["CursorBlinkRate"]);
        Console.WriteLine("Show content of windows when
              are draged." + mo["DragFullWindows"]);
        Console.WriteLine("Grid settings for dragging windows." +
              mo["GridGranularity"]);
        Console.WriteLine("Grid settings for icon spacing. " +
                       mo["IconSpacing"]);
        Console.WriteLine("Font used for the names of icons." +
                       mo["IconTitleFaceName"]);
        Console.WriteLine("Icon ront size. " + mo["IconTitleSize"]);
        Console.WriteLine("Wrapping of icon title." +
                       mo["IconTitleWrap"]);
        Console.WriteLine("Name of the desktop profile." +
                       mo["Name"]);
        Console.WriteLine("Screen saver is active." +
                       mo["ScreenSaverActive"]);
        Console.WriteLine("Name of the screen saver executable." +
                       mo["ScreenSaverExecutable"]);
        Console.WriteLine("Is screen saver protected
                       with password." + mo["ScreenSaverSecure"]);
        Console.WriteLine("Time to pass to activate screen saver." +
                       mo["ScreenSaverTimeout"]);
```

```
        Console.WriteLine("File name for desktop background." +
                          mo["Wallpaper"]);
        Console.WriteLine("Wallpaper fills entire screen." +
                          mo["WallpaperStretched"]);
        Console.WriteLine("Wallpaper is tiled." +
                          mo["WallpaperTiled"]);
      }
    }
  }
}
```

26.9. Определение типа компьютера

Запрос `Win32_ComputerSystem` используется для получения типа компьютера (листинг 26.8).

Листинг 26.8. Определение типа компьютера

```
using System;
using System.Management;

namespace Win32_ComputerSystem_1
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      string[] Roles = {
                 "Standalone Workstation",   // 0
                 "Member Workstation",       // 1
                 "Standalone Server",        // 2
                 "Member Server",            // 3
                 "Backup Domain Controller", // 4
                 "Primary Domain Controller" // 5
               };

      WqlObjectQuery query = new WqlObjectQuery(
                          "SELECT * FROM Win32_ComputerSystem");
      ManagementObjectSearcher find =
                          new ManagementObjectSearcher(query);
```

```
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine(Roles[Convert.ToInt32(mo["DomainRole"])]);
      }
    }
  }
}
```

26.10. Определение физических параметров компьютера

Листинг 26.9 демонстрирует получение физических параметров компьютера.

Листинг 26.9. Определение физических параметров компьютера

```
using System;
using System.Management;

namespace Win32_SystemEnclosure
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Перечисляем все модули компьютера
      WqlObjectQuery query = new WqlObjectQuery(
                      "SELECT * FROM Win32_SystemEnclosure");
      ManagementObjectSearcher find =
                      new ManagementObjectSearcher(query);
      int i = 0;
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("--- Chasis setting #" + i);
        Console.WriteLine("Chasis type." + GetChasisType(mo));
        Console.WriteLine("Description." + mo["Description"]);
        Console.WriteLine("Depth of physical package (in inches)." +
                mo["Depth"]);
        Console.WriteLine("Height of physical package
                 (in inches).." + mo["Height"]);
        Console.WriteLine("Width of physical package (in inches)." +
                mo["Width"]);
```

```
    Console.WriteLine("Weight." + mo["Weight"]);
    Console.WriteLine("Service philosophy " +
            GetServicePhilosophy(mo));
    Console.WriteLine("Status." + mo["Status"]);
    Console.WriteLine("Property includes visible alarm." +
            mo["VisibleAlarm"]);
    Console.WriteLine("Property includes visible alarm." +
            mo["VisibleAlarm"]);
    Console.WriteLine("-------------------------------");
    i++;
  }
}

private static string GetChasisType(ManagementObject mo)
{
  System.UInt16[] type = (System.UInt16[])mo["ChassisTypes"];
  String returnType = "";
  for (int i=0; i<type.Length; i++)
  {
    if (i > 0) returnType += ", ";
    switch (type[i])
    {
      case 1:
        returnType += "Other";
        break;
      case 2:
        returnType += "Unknown";
        break;
      case 3:
        returnType += "Desktop";
        break;
      case 4:
        returnType += "Low Profile Desktop";
        break;
      case 5:
        returnType += "Pizza Box";
        break;
      case 6:
        returnType += "Mini Tower";
        break;
      case 7:
        returnType += "Tower";
        break;
```

```
case 8:
  returnType += "Portable";
  break;
case 9:
  returnType += "Laptop";
  break;
case 10:
  returnType += "Notebook";
  break;
case 11:
  returnType += "Hand Held";
  break;
case 12:
  returnType += "Docking Station";
  break;
case 13:
  returnType += "All in One";
  break;
case 14:
  returnType += "Sub Notebook";
  break;
case 15:
  returnType += "Space-Saving";
  break;
case 16:
  returnType += "Lunch Box";
  break;
case 17:
  returnType += "Main System Chassis";
  break;
case 18:
  returnType += "Expansion Chassis";
  break;
case 19:
  returnType += "SubChassis";
  break;
case 20:
  returnType += "Bus Expansion Chassis";
  break;
case 21:
  returnType += "Peripheral Chassis";
  break;
```

```
          case 22:
            returnType += "Storage Chassis";
            break;
          case 23:
            returnType += "Rack Mount Chassis";
            break;
          case 24:
            returnType += "Sealed-Case PC";
            break;
        }
      }
      return returnType;
    }

    private static string GetServicePhilosophy(ManagementObject mo)
    {
      int i = Convert.ToInt16(mo["ServicePhilosophy"]);
      switch (i)
      {
        case 0:
          return "Unknown";
        case 1:
          return "Other";
        case 2:
          return "Service From Top";
        case 3:
          return "Service From Front";
        case 4:
          return "Service From Back";
        case 5:
          return "Service From Side";
        case 6:
          return "Sliding Trays";
        case 7:
          return "Removable Sides";
        case 8:
          return "Moveable";
      }
      return "Service philosophy not defined.";
    }
  }
}
```

26.11. Установка имени компьютера

Запрос `Win32_ComputerSystem` может использоваться для задания имени компьютера (листинг 26.10).

Листинг 26.10. Установка имени компьютера

```
using System;
using System.Management;

namespace Win32_ComputerSystem_2
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query = new WqlObjectQuery(
                 "SELECT * FROM Win32_ComputerSystem");
      ManagementObjectSearcher find =
                 new ManagementObjectSearcher(query);
      // Новое имя компьютера
      object[] methodArgs = {"New name"};
      foreach (ManagementObject mo in find.Get())
      {
        // Вызвать метод Rename
        mo.InvokeMethod("Rename", methodArgs);
      }
    }
  }
}
```

26.12. Получение информации о процессорах

Запрос `Win32_Processor` применяется для получения информации о процессорах (листинг 26.11).

Листинг 26.11. Получение информации о процессорах

```
using System;
using System.Management;
```

```
namespace Win32_Processor
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Получаем все процессоры компьютера
      WqlObjectQuery query = new WqlObjectQuery(
                      "Select * from Win32_Processor");
      ManagementObjectSearcher find =
                      new ManagementObjectSearcher(query);
      int i = 0;
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("-- Processor #" + i + " -");
        Console.WriteLine("Processor address width in bits." +
                 mo["AddressWidth"]);
        Console.WriteLine("Architecture of processor." +
                 GetArchitecture(mo));
        Console.WriteLine("Caption." + mo["Caption"]);
        Console.WriteLine("Processor address width in bits." +
                 mo["AddressWidth"]);
        Console.WriteLine("Usage status of the processor." +
                 GetCpuStatus(mo));
        Console.WriteLine("Current clock speed (in MHz)." +
                 mo["CurrentClockSpeed"]);
        Console.WriteLine("Processor data width." + mo["DataWidth"]);
        Console.WriteLine("Unique string identification." +
                 mo["DeviceID"]);
        Console.WriteLine("External clock frequency." +
                 mo["ExtClock"]);
        Console.WriteLine("Processor family." + GetFamily(mo));
        Console.WriteLine("L2 cache size." + mo["L2CacheSize"]);
        Console.WriteLine("L2 cache speed." + mo["L2CacheSpeed"]);
        Console.WriteLine("Load percentage (average
                value for second)." + mo["LoadPercentage"]);
        Console.WriteLine("Manufacturer." + mo["Manufacturer"]);
        Console.WriteLine("Maximum speed (in MHz)." +
                mo["MaxClockSpeed"]);
        Console.WriteLine("Name." + mo["Name"]);
        Console.WriteLine("Support for power management." +
                mo["PowerManagementSupported"]);
```

```
    Console.WriteLine("Unique identificator
            describing processor"+ mo["ProcessorId"]);
    Console.WriteLine("Processor type." + GetProcessorType(mo));
    Console.WriteLine("Role (CPU/math)." + mo["Role"]);
    Console.WriteLine("Socket designation." +
            mo["SocketDesignation"]);
    Console.WriteLine("Status." + mo["Status"]);
    Console.WriteLine("Status
            information." + GetStatusInfo(mo));
    Console.WriteLine("Processor version." + mo["Version"]);
    Console.WriteLine("Socket voltage." + mo["VoltageCaps"]);
    i++;
  }
}

private static string GetArchitecture(ManagementObject mo)
{
  int i = Convert.ToInt32(mo["Architecture"]);
  switch (i)
  {
    case 0:  return "x86";
    case 1:  return "MIPS";
    case 2:  return "Alpha";
    case 3:  return "PowerPC";
    case 4:  return "ia64";
  }
  return "Undefined";
}

private static string GetCpuStatus(ManagementObject mo)
{
  int i = Convert.ToInt32(mo["CpuStatus"]);
  switch(i)
  {
    case 0:  return "Unknown";
    case 1:  return "CPU Enabled";
    case 2:  return "CPU Disabled by User via BIOS Setup";
    case 3:  return "CPU Disabled By BIOS (POST Error)";
    case 4:  return "CPU is Idle";
    case 5:  return "This value is reserved.";
    case 6:  return "This value is reserved.";
    case 7:  return "Other";
  }
```

```
    return "Undefined";
  }

  private static string GetFamily(ManagementObject mo)
  {
    int i = Convert.ToInt32(mo["Family"]);
    switch (i)
    {
      case 1:  return "Other";
      case 2:  return "Unknown";
      case 3:  return "8086";
      case 4:  return "80286";
      case 5:  return "80386";
      case 6:  return "80486";
      case 7:  return "8087";
      case 8:  return "80287";
      case 9:  return "80387";
      case 10: return "80487";
      case 11: return "PentiumR brand";
      case 12: return "PentiumR Pro";
      case 13: return "PentiumR II";
      case 14: return "PentiumR processor with MMX technology";
      case 15: return "CeleronT";
      case 16: return "PentiumR II Xeon";
      case 17: return "PentiumR III";
      case 18: return "M1 Family";
      case 19: return "M2 Family";
      case 24: return "K5 Family";
      case 25: return "K6 Family";
      case 26: return "K6-2";
      case 27: return "K6-3";
      case 28: return "AMD AthlonT Processor Family";
      case 29: return "AMDR DuronT Processor";
      case 30: return "AMD2900 Family";
      case 31: return "K6-2+";
      case 32: return "Power PC Family";
      case 33: return "Power PC 601";
      case 34: return "Power PC 603";
      case 35: return "Power PC 603+";
      case 36: return "Power PC 604";
      case 37: return "Power PC 620";
      case 38: return "Power PC X704";
      case 39: return "Power PC 750";
```

```
case 48: return "Alpha Family";
case 49: return "Alpha 21064";
case 50: return "Alpha 21066";
case 51: return "Alpha 21164";
case 52: return "Alpha 21164PC";
case 53: return "Alpha 21164a";
case 54: return "Alpha 21264";
case 55: return "Alpha 21364";
case 64: return "MIPS Family";
case 65: return "MIPS R4000";
case 66: return "MIPS R4200";
case 67: return "MIPS R4400";
case 68: return "MIPS R4600";
case 69: return "MIPS R10000";
case 80: return "SPARC Family";
case 81: return "SuperSPARC";
case 82: return "microSPARC II";
case 83: return "microSPARC IIep";
case 84: return  "UltraSPARC";
case 85: return  "UltraSPARC II";
case 86: return  "UltraSPARC IIi";
case 87: return  "UltraSPARC III";
case 88: return  "UltraSPARC IIIi";
case 96: return  "68040";
case 97: return  "68xxx Family";
case 98: return  "68000";
case 99: return  "68010";
case 100: return  "68020";
case 101: return  "68030";
case 112: return  "Hobbit Family";
case 120: return  "CrusoeT TM5000 Family";
case 121: return  "CrusoeT TM3000 Family";
case 128: return  "Weitek";
case 130: return  "ItaniumT Processor";
case 144: return  "PA-RISC Family";
case 145: return  "PA-RISC 8500";
case 146: return  "PA-RISC 8000";
case 147: return  "PA-RISC 7300LC";
case 148: return  "PA-RISC 7200";
case 149: return  "PA-RISC 7100LC";
case 150: return  "PA-RISC 7100";
case 160: return  "V30 Family";
case 176: return  "PentiumR III XeonT";
```

```
      case 177: return  "PentiumR III Processor
                         with IntelR SpeedStepT Technology";
      case 178: return  "PentiumR 4";
      case 179: return  "IntelR XeonT";
      case 180: return  "AS400 Family";
      case 181: return  "IntelR XeonT processor MP";
      case 182: return  "AMD AthlonXPT Family";
      case 183: return  "AMD AthlonMPT Family";
      case 184: return  "IntelR ItaniumR 2";
      case 185: return  "AMD OpteronT Family";
      case 190: return  "K7";
      case 200: return  "IBM390 Family";
      case 201: return  "G4";
      case 202: return  "G5";
      case 250: return  "i860";
      case 251: return  "i960";
      case 260: return  "SH-3";
      case 261: return  "SH-4";
      case 280: return  "ARM";
      case 281: return  "StrongARM";
      case 300: return  "6x86";
      case 301: return  "MediaGX";
      case 302: return  "MII";
      case 320: return  "WinChip";
      case 350: return  "DSP";
      case 500: return  "Video Processor";
    }
    return "Undefined processor family";
  }

  private static string GetProcessorType(ManagementObject mo)
  {
    int i = Convert.ToInt32(mo["ProcessorType"]);
    switch (i)
    {
      case 1: return  "Other";
      case 2: return  "Unknown";
      case 3: return  "Central Processor";
      case 4: return  "Math Processor";
      case 5: return  "DSP Processor";
      case 6: return  "Video Processor";
    }
    return "Undefined type";
  }
```

```
    private static string GetStatusInfo(ManagementObject mo)
    {
      int i = Convert.ToInt32(mo["StatusInfo"]);
      switch (i)
      {
        case 1: return  "Other";
        case 2: return  "Unknown";
        case 3: return  "Enabled";
        case 4: return  "Disabled";
        case 5: return  "Not applicable";
      }
      return "StatusInfo not defined.";
    }
  }
}
```

26.13. Получение списка общих файлов и каталогов

Запрос `Win32_Share` используется для получения списка общих файлов и каталогов (листинг 26.12).

Листинг 26.12. Получение списка общих файлов и каталогов

```
using System;
using System.Management;

namespace Win32_Share
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query = new WqlObjectQuery(
                 "SELECT * FROM Win32_Share");
      ManagementObjectSearcher find =
                 new ManagementObjectSearcher(query);
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("List of shares = " + mo["Name"]);
      }
    }
  }
}
```

26.14. Создание и удаление общего каталога

С помощью объекта `Win32_Share` можно создать или удалить общий каталог (shared). Листинг 26.13 показывает пример создания общего каталога. Удаление каталога производится аналогично, с помощью вызова метода `Delete`.

Листинг 26.13. Создание общего каталога

```
using System;
using System.Management;

namespace Win32_Share_1
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      string folderName = @"C:\MyTestShare";
      string shareName  = @"My Files Share";

      try
      {
        // Создаем объект ManagementClass
        ManagementClass managementClass =
                              new ManagementClass("Win32_Share");
        // Создаем объект ManagementBaseObjects
        ManagementBaseObject inParams =
                    managementClass.GetMethodParameters("Create");
        ManagementBaseObject outParams;
        // Задаем параметры вызова
        inParams["Description"] = shareName;
        inParams["Name"] = shareName;
        inParams["Path"] = folderName;
        inParams["Type"] = 0x0;          // диск
        // Вызываем метод Create
        outParams = managementClass.InvokeMethod(
                              "Create", inParams, null);
        // Проверяем результат
        if((uint)(outParams.Properties["ReturnValue"].Value) != 0)
        {
          throw new Exception("Не удалось сделать каталог общим.");
        }
      }
```

```
      catch(Exception e)
      {
        Console.WriteLine(e.Message);
      }
    }
  }
}
```

26.15. Перечисление подключенных сетевых ресурсов

Перечисление подключенных сетевых ресурсов можно выполнить с помощью класса `Win32_NetworkConnection` (листинг 26.14).

Листинг 26.14. Перечисление подключенных сетевых ресурсов

```
using System;
using System.Management;

namespace Win32_NetworkConnection
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      ManagementClass managementClass =
                    new ManagementClass("Win32_NetworkConnection");
      ManagementObjectCollection managementObj =
                    managementClass.GetInstances();

      foreach(ManagementObject mo in managementObj)
      {
        Console.WriteLine("AccessMask:\t{0}", mo["AccessMask"]);
        Console.WriteLine("Caption:\t{0}", mo["Caption"]);
        Console.WriteLine("Comment:\t{0}", mo["Comment"]);
        Console.WriteLine("ConnectionState:\t{0}",
                                    mo["ConnectionState"]);
        Console.WriteLine("ConnectionType:\t{0}",
                                    mo["ConnectionType"]);
        Console.WriteLine("Description:\t{0}", mo["Description"]);
        Console.WriteLine("DisplayType:\t{0}", mo["DisplayType"]);
        Console.WriteLine("InstallDate:\t{0}", mo["InstallDate"]);
        Console.WriteLine("LocalName:\t{0}", mo["LocalName"]);
```

```
        Console.WriteLine("Name:\t{0}", mo["Name"]);
        Console.WriteLine("Persistent:\t{0}", mo["Persistent"]);
        Console.WriteLine("ProviderName:\t{0}", mo["ProviderName"]);
        Console.WriteLine("RemoteName:\t{0}", mo["RemoteName"]);
        Console.WriteLine("RemotePath:\t{0}", mo["RemotePath"]);
        Console.WriteLine("ResourceType:\t{0}", mo["ResourceType"]);
        Console.WriteLine("Status:\t{0}", mo["Status"]);
        Console.WriteLine("UserName:\t{0}", mo["UserName"]);
        Console.WriteLine("---------------------------------");
      }
    }
  }
}
```

26.16. Получение информации о диске

26.16.1. Вариант 1

Запрос `Win32_LogicalDisk` используется для получения информации о диске (листинг 26.15).

Листинг 26.15. Получение информации о диске

```
using System;
using System.Management;

namespace Win32_LogicalDisk
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      string cmiPath =
              @"\root\cimv2:Win32_LogicalDisk.DeviceID='C:'";
      ManagementObject mo = new ManagementObject(cmiPath);

      Console.WriteLine("Description: " + mo["Description"]);
      Console.WriteLine("File system: " + mo["FileSystem"]);
      Console.WriteLine("Free disk space: " + mo["FreeSpace"]);
      Console.WriteLine("Size: " + mo["Size"]);
    }
  }
}
```

26.16.2. Вариант 2

Запрос `Win32_LogicalDisk` используется для получения информации о диске (листинг 26.16).

Листинг 26.16. Получение информации о диске

```
using System;
using System.Management;

namespace Win32_LogicalDisk_1
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query = new WqlObjectQuery(
         "SELECT * FROM Win32_LogicalDisk WHERE DeviceID = 'C:'");
      ManagementObjectSearcher find =
          new ManagementObjectSearcher(query);
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("Description: " + mo["Description"]);
        Console.WriteLine("File system: " + mo["FileSystem"]);
        Console.WriteLine("Free disk space: " + mo["FreeSpace"]);
        Console.WriteLine("Size: " + mo["Size"]);
      }
    }
  }
}
```

26.17. Получение переменных окружения

Запрос `Win32_Environment` применяется для получения переменных окружения (листинг 26.17).

Листинг 26.17. Получение переменных окружения

```
using System;
using System.Management;

namespace Win32_Environment
{
  class Class1
```

```
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query = new WqlObjectQuery(
              "Select * from Win32_Environment");
      ManagementObjectSearcher find =
              new ManagementObjectSearcher(query);
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine(mo["Description"] + " - " + mo["Name"] +
           " - " + mo["UserName"] + " - " + mo["VariableValue"]);
      }
    }
  }
}
```

26.18. Получение информации о загрузчике системы

Запрос `Win32_BootConfiguration` используется для получения информации о загрузчике системы (листинг 26.18).

Листинг 26.18. Получение информации о загрузчике системы

```
using System;
using System.Management;

namespace Win32_BootConfiguration
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query = new WqlObjectQuery(
                "SELECT * FROM Win32_BootConfiguration");
      ManagementObjectSearcher find =
                new ManagementObjectSearcher(query);
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("Boot directory with files required
                    for booting." + mo["BootDirectory"]);
```

```
        Console.WriteLine("Description." + mo["Description"]);
        Console.WriteLine("Directory with temporary files
                    for booting." + mo["ScratchDirectory"]);
        Console.WriteLine("Directory with temporary files." +
                    mo["TempDirectory"]);
      }
    }
  }
}
```

26.19. Получение списка процессов

Запрос `Win32_Process` используется для получения списка процессов (листинг 26.19).

Листинг 26.19. Получение списка процессов

```
using System;
using System.Management;

namespace Win32_Process
{
  class Class1
  {
    const string Query = "SELECT * FROM Win32_Process";
    const int TextSize = 256;
    enum ResultCode:uint
    {
      SuccessfullCompletion = 0,
      AccessDenied          = 2,
      InsufficientPrivilege = 3,
      UnknownFailure        = 4,
      PathNotFound          = 9,
      InvalidParameter      = 21
    }

    [STAThread]
    static void Main(string[] args)
    {
      ManagementObjectSearcher processEnumerator =
                new ManagementObjectSearcher(Query);
      foreach(ManagementObject process in processEnumerator.Get())
      {
        Console.WriteLine("Процесс: " + process["Name"] as string);
```

```
        object[] parameters = new object[2];
        ResultCode result =
          (ResultCode)process.InvokeMethod("GetOwner", parameters);
        if (result == ResultCode.SuccessfullCompletion)
        {
          Console.WriteLine("Пользователь: " + parameters[0]);
          Console.WriteLine("Домен: " + parameters[1]);
        }
        else
        {
          Console.Write("Ошибка: " + result);
        }
        Console.WriteLine(Environment.NewLine);
      }
    }
  }
}
```

26.20. Получение списка запущенных и остановленных сервисов

Запрос `Win32_Service` используется для получения списка сервисов (листинг 26.20).

Листинг 26.20. Получение списка запущенных и остановленных сервисов

```
using System;
using System.Management;

namespace Win32_Process_1
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Список запущенных сервисов
      WqlObjectQuery query_run = new WqlObjectQuery(
              "SELECT * FROM Win32_Service WHERE state='running'");
      ManagementObjectSearcher find_run =
                      new ManagementObjectSearcher(query_run);
```

```
      foreach (ManagementObject mo in find_run.Get())
      {
        Console.WriteLine(
                  "Имя сервиса    : " + mo["DisplayName"] +
                  "- Режим        : " + mo["StartMode"]   +
                  "- Описание     : " + mo["Description"]
        );
      }

      Console.WriteLine(new string('-', 20));

      // Список остановленных сервисов
      WqlObjectQuery query_stp = new WqlObjectQuery(
              "SELECT * FROM Win32_Service WHERE state='stopped'");
      ManagementObjectSearcher find_stp =
              new ManagementObjectSearcher(query_stp);

      foreach (ManagementObject mo in find_stp.Get())
      {
        Console.WriteLine(
                "Сервис     : " + mo["DisplayName"] +
                "-- Режим   : " + mo["StartMode"]   +
                "-- Описание: " + mo["Description"]
        );
      }
    }
  }
}
```

26.21. Получение информации о разделах диска

Запрос `Win32_DiskPartition` используется для получения информации о разделах диска (листинг 26.21).

Листинг 26.21. Получение информации о разделах (partition info)

```
using System;
using System.Management;

namespace Win32_DiskPartition
{
  class Class1
```

```
{
  [STAThread]
  static void Main(string[] args)
  {
    WqlObjectQuery query = new WqlObjectQuery(
                    "Select * from Win32_DiskPartition");
    ManagementObjectSearcher find =
                    new ManagementObjectSearcher(query);
    foreach (ManagementObject mo in find.Get())
    {
      Console.WriteLine("Block size." +
                mo["BlockSize"] + " Bytes");
      Console.WriteLine("Partition is labeled as bootable. " +
                  mo["Bootable"]);
      Console.WriteLine("Boot partition active. " +
                  mo["BootPartition"]);
      Console.WriteLine("Caption.." + mo["Caption"]);
      Console.WriteLine("Description." + mo["Description"]);
      Console.WriteLine("Unique identification of partition.." +
                  mo["DeviceID"]);
      Console.WriteLine("Index number of the
                  disk with that partition." + mo["DiskIndex"]);
      Console.WriteLine("Detailed description of error
                  in LastErrorCode." + mo["ErrorDescription"]);
      Console.WriteLine("Type of error detection
                  and correction." + mo["ErrorMethodology"]);
      Console.WriteLine("Hidden sectors in partition." +
                  mo["HiddenSectors"]);
      Console.WriteLine("Index number of the partition." +
                  mo["Index"]);
      Console.WriteLine("Last error by device." +
                  mo["LastErrorCode"]);
      Console.WriteLine("Total number of consecutive blocks." +
                  mo["NumberOfBlocks"]);
      Console.WriteLine("Partition labeled as primary." +
                  mo["PrimaryPartition"]);
      Console.WriteLine("Free description of media purpose. " +
                  mo["Purpose"]);
      Console.WriteLine("Total size of partition." +
                  mo["Size"] + " bytes");
      Console.WriteLine("Starting offset of the partition " +
                  mo["StartingOffset"]);
```

```
        Console.WriteLine("Status." + mo["Status"]);
        Console.WriteLine("Type of the partition." + mo["Type"]);
      }
    }
  }
}
```

26.22. Получение учетных записей локальной машины или домена

Запрос `Win32_UserAccount` применяется для получения учетных записей локальной машины или домена (листинг 26.22).

Листинг 26.22. Получение учетных записей локальной машины или домена

```
using System;
using System.Management;

namespace Win32_UserAccount
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Выбираем все локальные группы, не группы домена.
      // Можно заменить условие LocalAccount, например,
      // на условие Domain = 'domain_name'
      WqlObjectQuery query = new WqlObjectQuery(
       "SELECT * FROM Win32_UserAccount
                    WHERE LocalAccount =  'true'");
      ManagementObjectSearcher find =
        new ManagementObjectSearcher(query);
      Console.WriteLine("------------");
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("Caption." + mo["Caption"]);
        Console.WriteLine("Description." + mo["Description"]);
        Console.WriteLine("Domain where account belongs." +
                  mo["Domain"]);
        Console.WriteLine("Account is defined on local machine. " +
                  mo["LocalAccount"]);
```

```
        Console.WriteLine("Name of the account " + mo["Name"]);
        Console.WriteLine("Password can be changed." +
                  mo["PasswordChangeable"]);
        Console.WriteLine("Password expires." +
                  mo["PasswordExpires"]);
        Console.WriteLine("Password is required for this account." +
                  mo["PasswordRequired"]);
        Console.WriteLine("Security identifier (SID)." + mo["SID"]);
        Console.WriteLine("Type of security identifier." +
                  GetSidType(Convert.ToInt32(mo["SIDType"])));
        Console.WriteLine("Status." + mo["Status"]);
        Console.WriteLine("-------------");
      }
    }

    public static string GetSidType(int type)
    {
      switch (type)
      {
        case 1: return "SidTypeUser";
        case 2: return "SidTypeGroup";
        case 3: return "SidTypeDomain";
        case 4: return "SidTypeAlias";
        case 5: return "SidTypeWellKnownGroup";
        case 6: return "SidTypeDeletedAccount";
        case 7: return "SidTypeInvalid";
        case 8: return "SidTypeUnknown";
        case 9: return "SidTypeComputer";
      }
      return string.Empty;
    }
  }
}
```

26.23. Получение списка групп локальной машины или домена

Запрос `Win32_Group` используется для получения списка групп локальной машины или домена (листинг 26.23). Если нужно получить список групп домена, в запросе следует заменить условие `LocalAccount = 'true'` на `Domain = 'domain_name'`.

Листинг 26.23. Получение списка групп локальной машины или домена

```
using System;
using System.Management;

namespace Win32_Group
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      WqlObjectQuery query = new WqlObjectQuery(
         "Select * from Win32_Group where LocalAccount = 'true'");
      ManagementObjectSearcher find =
         new ManagementObjectSearcher(query);
      Console.WriteLine("-------------");
      foreach (ManagementObject mo in find.Get())
      {
        Console.WriteLine("Caption." + mo["Caption"]);
        Console.WriteLine("Description." + mo["Description"]);
        Console.WriteLine("Domain where group belongs." +
                    mo["Domain"]);
        Console.WriteLine("Account is defined on local machine." +
                    mo["LocalAccount"]);
        Console.WriteLine("Name of the group." + mo["Name"]);
        Console.WriteLine("Security identifier (SID)." + mo["SID"]);
        Console.WriteLine("Type of security identifier. " +
                   GetSidType(Convert.ToInt32(mo["SIDType"])));
        Console.WriteLine("Status." + mo["Status"]);
        Console.WriteLine("------------");
      }
    }

    public static string GetSidType(int type)
    {
      switch (type)
      {
        case 1:  return "SidTypeUser";
        case 2:  return "SidTypeGroup";
        case 3:  return "SidTypeDomain";
        case 4:  return "SidTypeAlias";
        case 5:  return "SidTypeWellKnownGroup";
        case 6:  return "SidTypeDeletedAccount";
```

```
        case 7:  return "SidTypeInvalid";
        case 8:  return "SidTypeUnknown";
        case 9:  return "SidTypeComputer";
      }
      return string.Empty;
    }
  }
}
```

ГЛАВА 27

Работа с аппаратурой

27.1. Импортирование функций Setup API

Для работы со списком устройств требуется импортирование функций модуля Setup API [1] (листинг 27.1).

Листинг 27.1. Функции Setup API

```
namespace DeviceEnumerator
{
  using System;
  using System.Runtime.InteropServices;

  public class SetupAPI
  {
    [DllImport("hid.dll", SetLastError=true)]
    public static extern unsafe void HidD_GetHidGuid(
      ref Guid lpHidGuid
      );

    [DllImport("setupapi.dll", SetLastError=true)]
    public static extern unsafe int SetupDiGetClassDevs (
      ref Guid lpGuid,
      int*  Enumerator,
      int*  hwndParent,
      ClassDevsFlags Flags
      );

    [DllImport("setupapi.dll", SetLastError=true)]
    public static extern unsafe int SetupDiGetClassDevs (
      int* guid,
```

```
  int* Enumerator,
  int* hwndParent,
  ClassDevsFlags Flags
  );

[StructLayout(LayoutKind.Sequential)]
public struct SP_DEVINFO_DATA
{
  public int cbSize;
  public Guid ClassGuid;
  public int DevInst;
  public int Reserved;
}

[DllImport("setupapi.dll", SetLastError=true)]
public static extern unsafe int SetupDiEnumDeviceInfo(
  int DeviceInfoSet,
  int  Index,
  ref SP_DEVINFO_DATA DeviceInfoData
  );

[Flags]
public enum ClassDevsFlags
{
  DIGCF_DEFAULT         = 0x00000001,
  DIGCF_PRESENT         = 0x00000002,
  DIGCF_ALLCLASSES      = 0x00000004,
  DIGCF_PROFILE         = 0x00000008,
  DIGCF_DEVICEINTERFACE = 0x00000010,
}

// Данные интерфейса устройства
[StructLayout(LayoutKind.Sequential)]
public unsafe struct SP_DEVICE_INTERFACE_DATA
{
  public int cbSize;
  public Guid InterfaceClassGuid;
  public int Flags;
  public int Reserved;
}

// Подробные данные интерфейса устройства
[StructLayout(LayoutKind.Sequential, CharSet= CharSet.Ansi)]
```

```
public unsafe struct PSP_DEVICE_INTERFACE_DETAIL_DATA
{
  public int cbSize;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst= 256)]
  public string DevicePath;
}

[DllImport("setupapi.dll", SetLastError=true)]
public static extern unsafe int SetupDiEnumDeviceInterfaces (
  int DeviceInfoSet,
  int DeviceInfoData,
  ref Guid lpHidGuid,
  int MemberIndex,
  ref SP_DEVICE_INTERFACE_DATA lpDeviceInterfaceData);

[DllImport("setupapi.dll", SetLastError=true)]
public static extern unsafe int SetupDiGetDeviceInterfaceDetail (
  int DeviceInfoSet,
  ref SP_DEVICE_INTERFACE_DATA lpDeviceInterfaceData,
  int* aPtr,
  int  detailSize,
  ref  int requiredSize,
  int* bPtr);

[DllImport("setupapi.dll", SetLastError=true)]
public static extern unsafe int SetupDiGetDeviceInterfaceDetail (
  int DeviceInfoSet,
  ref SP_DEVICE_INTERFACE_DATA lpDeviceInterfaceData,
  ref PSP_DEVICE_INTERFACE_DETAIL_DATA
                          myPSP_DEVICE_INTERFACE_DETAIL_DATA,
  int     detailSize,
  ref int requiredSize,
  int* bPtr);

// Полный список флагов можно найти в исходном коде
public enum RegPropertyType
{
  SPDRP_DEVICEDESC    = 0x00000000, // DeviceDesc (R/W)
  SPDRP_HARDWAREID    = 0x00000001, // HardwareID (R/W)
  SPDRP_DRIVER        = 0x00000009, // Driver (R/W)
}

[DllImport("setupapi.dll", SetLastError=true)]
public static extern unsafe
```

```
     int SetupDiGetDeviceRegistryProperty (
      int                 DeviceInfoSet,
      ref SP_DEVINFO_DATA DeviceInfoData,
      RegPropertyType     Property,
      int*                PropertyRegDataType,
      int*                PropertyBuffer,
      int                 PropertyBufferSize,
      ref int             RequiredSize
    );

    // Подробные данные интерфейса устройства
    [StructLayout(LayoutKind.Sequential, CharSet= CharSet.Ansi)]
    public unsafe struct DATA_BUFFER
    {
      [MarshalAs(UnmanagedType.ByValTStr, SizeConst= 1024)]
      public string Buffer;
    }

    [DllImport("setupapi.dll", SetLastError=true)]
    public static extern unsafe
                        int SetupDiGetDeviceRegistryProperty (
      int                 DeviceInfoSet,
      ref SP_DEVINFO_DATA DeviceInfoData,
      RegPropertyType     Property,
      int*                PropertyRegDataType,
      ref DATA_BUFFER     PropertyBuffer,
      int                 PropertyBufferSize,
      ref int             RequiredSize
    );
  }
}
```

27.2. Перечисление устройств

Для перечисления устройств (листинг 27.2) и получения их свойств (листинг 27.3) используются функции библиотеки Setup API, которые необходимо импортировать (см. листинг 27.1) [1].

Листинг 27.2. Перечисление устройств

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

```
namespace DeviceEnumerator1
{
  class Class1
  {
    [STAThread]
    static unsafe void Main(string[] args)
    {
      int PnPHandle = SetupAPI.SetupDiGetClassDevs(
        null,
        null,
        null,
        SetupAPI.ClassDevsFlags.DIGCF_ALLCLASSES |
                        SetupAPI.ClassDevsFlags.DIGCF_PRESENT
      );
      Console.WriteLine("PnPHandle={0}", PnPHandle);

      int  result   = -1;
      int DeviceIndex = 0;

      while (result != 0)
      {
        SetupAPI.SP_DEVINFO_DATA DeviceInfoData =
                               new SetupAPI.SP_DEVINFO_DATA();
        DeviceInfoData.cbSize = Marshal.SizeOf(DeviceInfoData);

        result = SetupAPI.SetupDiEnumDeviceInfo(PnPHandle,
                         DeviceIndex, ref DeviceInfoData);

        if (result == 1)
        {
          int RequiredSize = 0;
          SetupAPI.DATA_BUFFER Buffer = new SetupAPI.DATA_BUFFER();
          result = SetupAPI.SetupDiGetDeviceRegistryProperty(
            PnPHandle,
            ref DeviceInfoData,
            SetupAPI.RegPropertyType.SPDRP_DEVICEDESC,
            null,
            ref Buffer,
            1024,
            ref RequiredSize
            );
          Console.WriteLine("{0}", Buffer.Buffer);
        }
```

```
        DeviceIndex++;
      }
    }
  }
}
```

Листинг 27.3. Получение свойств устройств

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

namespace DeviceEnumerator1
{
  class Class1
  {
    [STAThread]
    static unsafe void Main(string[] args)
    {
      // DisplayGuid : TGUID =
      //           '{4d36e968-e325-11ce-bfc1-08002be10318}';
      // HidGuid  : TGUID =
      //           '{745a17a0-74d3-11d0-b6fe-00a0c90f57da}';
      // USBGuid  : TGUID =
      //           '{36FC9E60-C465-11CF-8056-444553540000}';
      Guid guid =
            new Guid("{36FC9E60-C465-11CF-8056-444553540000}");

      int PnPHandle = SetupAPI.SetupDiGetClassDevs(
        ref guid,
        null,
        null,
        SetupAPI.ClassDevsFlags.DIGCF_PRESENT
      );

      int  result   = -1;
      int DeviceIndex = 0;

      while (result != 0)
      {
        SetupAPI.SP_DEVINFO_DATA DeviceInfoData =
                        new SetupAPI.SP_DEVINFO_DATA();
        DeviceInfoData.cbSize = Marshal.SizeOf(DeviceInfoData);
        result = SetupAPI.SetupDiEnumDeviceInfo(PnPHandle,
                        DeviceIndex, ref DeviceInfoData);
```

```
        if (result == 1)
        {
          Console.WriteLine("{0}:\n\t{1}\n\t{2}\n\t{3}\n\t{4}",
            GetRegistryProperty(PnPHandle, ref DeviceInfoData,
                      SetupAPI.RegPropertyType.SPDRP_DEVICEDESC),
            GetRegistryProperty(PnPHandle, ref DeviceInfoData,
                      SetupAPI.RegPropertyType.SPDRP_CLASS),
            GetRegistryProperty(PnPHandle, ref DeviceInfoData,
                      SetupAPI.RegPropertyType.SPDRP_CLASSGUID),
            GetRegistryProperty(PnPHandle, ref DeviceInfoData,
                      SetupAPI.RegPropertyType.SPDRP_DRIVER),
            GetRegistryProperty(PnPHandle, ref DeviceInfoData,
                      SetupAPI.RegPropertyType.SPDRP_MFG   )
          );
        }
        DeviceIndex++;
      }
    }

    public unsafe static string GetRegistryProperty(int PnPHandle,
          ref SetupAPI.SP_DEVINFO_DATA DeviceInfoData,
          SetupAPI.RegPropertyType Property)
    {
      int RequiredSize = 0;
      SetupAPI.DATA_BUFFER Buffer = new SetupAPI.DATA_BUFFER();

      int result = SetupAPI.SetupDiGetDeviceRegistryProperty(
        PnPHandle,
        ref DeviceInfoData,
        Property,
        null,
        ref Buffer,
        1024,
        ref RequiredSize
        );
      return Buffer.Buffer;
    }
  }
}
```

27.3. Получение состояния устройства

Метод `IsEnable`, реализованный в листинге 27.4, возвращает `true`, если устройство включено и функционирует корректно. Метод `IsDisableable` возвращает `true`, если устройство можно отключать.

Листинг 27.4. Получение состояния устройства

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

namespace DeviceStatus
{
  class Class1
  {
    [STAThread]
    static unsafe void Main(string[] args)
    {
      Guid UsbGuid =
           new Guid("{36FC9E60-C465-11CF-8056-444553540000}");

      int PnPHandle = SetupAPI.SetupDiGetClassDevs(
        ref UsbGuid,
        null,
        null,
        SetupAPI.ClassDevsFlags.DIGCF_PRESENT
      );

      int  result   = -1;
      int DeviceIndex = 0;

      while (result != 0)
      {
        SetupAPI.SP_DEVINFO_DATA DeviceInfoData =
                new SetupAPI.SP_DEVINFO_DATA();
        DeviceInfoData.cbSize = Marshal.SizeOf(DeviceInfoData);
        result = SetupAPI.SetupDiEnumDeviceInfo(PnPHandle,
                          DeviceIndex, ref DeviceInfoData);

        if (result == 1)
        {
          Console.WriteLine("{0}:\n\tIsEnabled={1}
                     \n\tIsDisableable={2}",
                    GetRegistryProperty(PnPHandle,
                    ref DeviceInfoData,
                    SetupAPI.RegPropertyType.SPDRP_DEVICEDESC),
                    IsEnable(DeviceInfoData),
                    IsDisableable(DeviceInfoData)
          );
        }
```

```
    DeviceIndex++;
  }
  Marshal.FreeHGlobal((System.IntPtr)PnPHandle);
}

public unsafe static string GetRegistryProperty(int PnPHandle,
                  ref SetupAPI.SP_DEVINFO_DATA DeviceInfoData,
                  SetupAPI.RegPropertyType Property)
{
  int RequiredSize = 0;
  SetupAPI.DATA_BUFFER Buffer = new SetupAPI.DATA_BUFFER();

  int result = SetupAPI.SetupDiGetDeviceRegistryProperty(
    PnPHandle,
    ref DeviceInfoData,
    Property,
    null,
    ref Buffer,
    1024,
    ref RequiredSize
    );
  return Buffer.Buffer;
}

public unsafe static bool IsEnable(
                   SetupAPI.SP_DEVINFO_DATA DevData)
{
  int Status  = 0;
  int Problem  = 0;

  SetupAPI.CM_Get_DevNode_Status(ref Status,
                  ref Problem, DevData.DevInst, 0);
  return !(((Status & SetupAPI.DN_HAS_PROBLEM)!=0) &&
                   (Problem == SetupAPI.CM_PROB_DISABLED));
}

public unsafe static bool IsDisableable(
                   SetupAPI.SP_DEVINFO_DATA DevData)
{
  int Status  = 0;
  int Problem  = 0;

  SetupAPI.CM_Get_DevNode_Status(ref Status,
                  ref Problem, DevData.DevInst, 0);
```

```
      return ((Status & SetupAPI.DN_DISABLEABLE)!=0) &&
               (Problem != SetupAPI.CM_PROB_HARDWARE_DISABLED);
    }
  }
}
```

27.4. Программное отключение устройства

Для программного отключения устройства (например, Flash-диска) используются функции Setup API (листинг 27.5).

Листинг 27.5. Программное отключение устройства

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

namespace DeviceEject
{
  class Class1
  {
    [STAThread]
    static unsafe void Main(string[] args)
    {
      Guid UsbGuid =
             new Guid("{36FC9E60-C465-11CF-8056-444553540000}");

      int PnPHandle = SetupAPI.SetupDiGetClassDevs(
        ref UsbGuid,
        null,
        null,
        SetupAPI.ClassDevsFlags.DIGCF_PRESENT
      );

      int  result     = -1;
      int DeviceIndex =  0;

      while (result != 0)
      {
        SetupAPI.SP_DEVINFO_DATA DeviceInfoData =
                                  new SetupAPI.SP_DEVINFO_DATA();
        DeviceInfoData.cbSize = Marshal.SizeOf(DeviceInfoData);
        result = SetupAPI.SetupDiEnumDeviceInfo(PnPHandle,
                                 DeviceIndex, ref DeviceInfoData);
```

```
    if (result == 1)
    {
      if (IsRemovable(DeviceInfoData))
      {
        Console.WriteLine("{0}", GetRegistryProperty(PnPHandle,
                  ref DeviceInfoData,
                  SetupAPI.RegPropertyType.SPDRP_DEVICEDESC));
        if (SetupAPI.CM_Request_Device_Eject(
                        DeviceInfoData.DevInst,
                        null, null, 0, 0) == 0)
          Console.WriteLine("Устройство успешно отключено.");
      }
    }
    DeviceIndex++;
  }
  Marshal.FreeHGlobal((System.IntPtr)PnPHandle);
}

public unsafe static string GetRegistryProperty(int PnPHandle,
                ref SetupAPI.SP_DEVINFO_DATA DeviceInfoData,
                SetupAPI.RegPropertyType Property)
{
  int RequiredSize = 0;
  SetupAPI.DATA_BUFFER Buffer = new SetupAPI.DATA_BUFFER();

  int result = SetupAPI.SetupDiGetDeviceRegistryProperty(
    PnPHandle,
    ref DeviceInfoData,
    Property,
    null,
    ref Buffer,
    1024,
    ref RequiredSize
    );
  return Buffer.Buffer;
}

public unsafe static bool IsRemovable(
                  SetupAPI.SP_DEVINFO_DATA DevData)
{
  int Status  = 0;
  int Problem  = 0;
  SetupAPI.CM_Get_DevNode_Status(ref Status,
                          ref Problem, DevData.DevInst, 0);
```

```
      return ((Status & SetupAPI.DN_REMOVABLE)!=0);
    }
  }
}
```

27.5. Отслеживание изменения аппаратной конфигурации

При изменении в конфигурации операционная система посылает сообщение WM_DEVICECHANGE (листинг 27.6).

Листинг 27.6. Обработка сообщения WM_DEVICECHANGE

```
const int WM_DEVICECHANGE  = 0x0219;
const int DBT_DEVICEARRIVAL  = 0x8000;
const int DBT_DEVICEREMOVECOMPLETE = 0x8004;

[StructLayout(LayoutKind.Sequential)]
public struct DEV_BROADCAST_HDR
{
  public int dbch_size;
  public int dbch_devicetype;
  public int dbch_reserved;
}

protected override void WndProc(ref Message m)
{
  if (m.Msg == WM_DEVICECHANGE)
  {
  int EventCode = m.WParam.ToInt32();
  Log(string.Format("WM_DEVICECHANGE. Код={0}", EventCode));

  switch (EventCode)
  {
   case DBT_DEVICEARRIVAL:
   {
     Log("Добавление устройства");
     break;
   }
   case DBT_DEVICEREMOVECOMPLETE:
   {
     Log("Удаление устройства");
```

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

}

private void Log(string s)
{
  listBox1.Items.Add(s);
  listBox1.SelectedIndex = listBox1.Items.Count-1;
}
```

27.6. Перечисление устройств ввода с помощью функций Direct X

Модуль Direct Input библиотеки Direct X позволяет перечислить все устройства ввода (мышь, джойстик, HID-устройства и т. д.). Пример использования этой библиотеки приведен в листинге 27.7.

Листинг 27.7. Перечисление устройств ввода с помощью функций DirectX

```
using System;
using Microsoft.DirectX;
using Microsoft.DirectX.DirectInput;

namespace USB.DirectX.HIDDeviceList
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {

      // Получаем DeviceList
      DeviceList dl = Microsoft.DirectX.DirectInput.Manager.Devices;
      Console.WriteLine("Всего устройств: {0}", dl.Count);

      // Для мыши выводит:
      //    type=Mouse, subtype=1, usage=page=0
      // Для клавиатуры выводит
      //    type=Keyboard, subtype=4, usage=page=0
```

```
      // Для HID выводит:
     //     type=Device, subtype=0, usage=1, page=65280

      foreach (DeviceInstance d in dl)
      {
        Console.WriteLine(
                 "\n\nType={0} SubType={1} Usage={2} Page={3}
                                    Guid={4} ProductName={5}",
          d.DeviceType,
          d.DeviceSubType,
          (UInt16)d.Usage,
          (UInt16)d.UsagePage,
          d.InstanceGuid,
          d.ProductName
          );
      }
    }
  }
}
```

27.7. Чтение скан-кодов клавиатуры с помощью функций Direct X

Модуль Direct Input библиотеки Direct X позволяет читать скан-коды клавиатуры напрямую с оборудования (листинг 27.8).

Листинг 27.8. Чтение скан-кодов клавиатуры с помощью функций DirectX

```
using System;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.DirectX.DirectInput;

public class MainForm : System.Windows.Forms.Form
{
... ... ... ... ... ... ... ... ... ...
  private Device device = null;
  private BufferedDataCollection dataCollection = null;

  public MainForm()
  {
    InitializeComponent();
```

```
    // Создаем объект устройства
    device = new Device(SystemGuid.Keyboard);
    // Режим работы
    device.SetCooperativeLevel(this,
           CooperativeLevelFlags.Background |
           CooperativeLevelFlags.NonExclusive);
    // Размер буфера
    device.Properties.BufferSize = 11;
    // Начало обмена
    device.Acquire();
  }

  private void btnRead_Click(object sender, System.EventArgs e)
  {
    string result = String.Empty;
    // Получаем буфер данных
    dataCollection = device.GetBufferedData();

    // Если есть данные - отображаем
    if(dataCollection != null)
    {
      foreach (BufferedData d in dataCollection)
      {
        // Выводим смещение и код
        result += String.Format(" 0x{0:X2}=0x{1:X2} ",
                                d.Offset, d.Data);
      }
    }
    tbResult.Text = result;
  }

}
```

27.8. Чтение данных с последовательного порта

27.8.1. Вариант 1

Для чтения данных с последовательного порта можно использовать ActiveX-компонент `MSComm` (листинг 27.9).

Листинг 20.9. Чтение данных с последовательного порта с помощью `MSComm`

```
using System;
using System.Drawing;
```

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

namespace MSCOMMTest
{
  public class Form1 : System.Windows.Forms.Form
  {
    private AxMSCommLib.AxMSComm axMSComm1;
    private System.Windows.Forms.ListBox lbLog;
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
    private void InitializeComponent()
    {
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
      this.axMSComm1 = new AxMSCommLib.AxMSComm();
      ((System.ComponentModel.ISupportInitialize)
                                    (this.axMSComm1)).BeginInit();
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
      this.axMSComm1.Enabled = true;
      this.axMSComm1.Location = new System.Drawing.Point(160, 72);
      this.axMSComm1.Name = "axMSComm1";
      this.axMSComm1.OcxState =
             ((System.Windows.Forms.AxHost.State)(
                    resources.GetObject("axMSComm1.OcxState")));
      this.axMSComm1.Size = new System.Drawing.Size(38, 38);
      this.axMSComm1.TabIndex = 0;
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
      ((System.ComponentModel.ISupportInitialize)
                                      (this.axMSComm1)).EndInit();
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
    void InitPort()
    {
      // Задаем номер порта
      axMSComm1.CommPort = 1;
       // Закрыть порт, если он открыт
      if (axMSComm1.PortOpen) axMSComm1.PortOpen = false;
       // Trigger the OnComm event whenever data is received
      axMSComm1.RThreshold = 1;
       // Настройка порта: 9600, no parity, 8 bits, 1 stop bit
      axMSComm1.Settings = "9600,n,8,1";
      // Режим приема данных - бинарный или текстовый
      axMSComm1.InputMode =
                  MSCommLib.InputModeConstants.comInputModeBinary;
```

```
  //axMSComm1.InputMode =
              MSCommLib.InputModeConstants.comInputModeText;
   // Режим ожидания данных
  axMSComm1.InputLen = 0;
  // Не игнорировать 0x00
  axMSComm1.NullDiscard = false;
  // Добавляем обработчик получения данных
  axMSComm1.OnComm += new System.EventHandler(this.OnComm);
  // Открываем порт
  axMSComm1.PortOpen = true;
}

// Обработчик получения данных
private void OnComm(object sender, EventArgs e)
{
  Application.DoEvents();

  // Обработка изменения состояния линии CTS
  if (axMSComm1.CommEvent ==
                    (short)MSCommLib.OnCommConstants.comEvCTS)
  {
    Log("Изменение состояния линии CTS");
  }

  // Если есть данные в буфере
  if (axMSComm1.InBufferCount > 0)
  {
    // Если включен режим comInputModeText,
    // мы получим строку
    if (axMSComm1.Input is String)
    {
      string Input = (string) axMSComm1.Input;
      Log(Input);
    }
    // Если включен режим comInputModeBinary, мы получим
    // System.Array
    else
    {
      byte [] Input = (byte[])axMSComm1.Input;
      foreach (byte b in  Input)
      {
        Log(string.Format("{0}", b));
      }
```

```
        }
      }
    }
    private void Log(string str)
    {
      lbLog.Items.Add(str);
      lbLog.SelectedIndex = lbLog.Items.Count-1;
    }
  }
}
```

27.8.2. Вариант 2

Чтение данных можно реализовать с помощью импортированных функций Win32 (листинги 27.10 и 27.11).

Листинг 27.10. Чтение данных с последовательного порта с помощью Win32-функций

```
private void button1_Click(object sender, System.EventArgs e)
{
  CommPort port = new CommPort(1, 9600, 8, Parity.No, StopBits.Bits1);
  if (port.Open())
  {
    byte[] data = port.Read(100);
    foreach (byte b in data)
    {
       listBox1.Items.Add(string.Format("{0}", b));
    }
    port.Close();
  }
 }
```

Листинг 27.11. Импортирование Win32-функций для последовательного порта

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

namespace ComPortRead
{
  public enum Parity : byte
  {
    No    = 0,
    Odd   = 1,
```

```
  Even  = 2,
  Mark  = 3,
  Space = 4
}

public enum StopBits : byte
{
  Bits1   = 0,
  Bits1_5 = 1,
  Bits2   = 2
}

class CommPort
{
  private int PortNum;
  private int BaudRate;
  private byte ByteSize;
  private Parity parity;
  private StopBits stopBits;
  private int hPortHanle = INVALID_HANDLE_VALUE;

  public CommPort(int PortNum, int BaudRate, byte ByteSize,
                  Parity parity, StopBits stopBits)
  {
    this.PortNum   = PortNum;
    this.BaudRate  = BaudRate;
    this.ByteSize  = ByteSize;
    this.parity    = parity;
    this.stopBits  = stopBits;
  }


  public bool Open()
  {
    // Открытие порта
    hPortHanle = CreateFile("COM" + PortNum ,GENERIC_READ |
                      GENERIC_WRITE,0, 0,OPEN_EXISTING,0,0);
    if(hPortHanle == INVALID_HANDLE_VALUE)
    {
      return  false;
    }

    // Настройка порта
    DCB dcbCommPort = new DCB();
```

```
  GetCommState(hPortHanle, ref dcbCommPort);
  dcbCommPort.BaudRate = BaudRate;
  dcbCommPort.Parity = (byte)parity;
  dcbCommPort.ByteSize = ByteSize;
  dcbCommPort.StopBits = (byte)stopBits;
  if (!SetCommState(hPortHanle, ref dcbCommPort))
  {
     return  false;
  }
  return true;
}

// Возвращает true, если порт открыт
public bool IsOpen()
{
  return(hPortHanle!=INVALID_HANDLE_VALUE);
}
// Закрытие порта
public void Close()
{
  if (IsOpen())
  {
    CloseHandle(hPortHanle);
  }
}

// Чтение данных
public byte[] Read(int NumBytes)
{
  byte[] BufBytes;
  byte[] OutBytes;
  BufBytes = new byte[NumBytes];
  if (hPortHanle!=INVALID_HANDLE_VALUE)
  {
    int BytesRead=0;
    ReadFile(hPortHanle, BufBytes, NumBytes, ref BytesRead, 0);
    OutBytes = new byte[BytesRead];
    Array.Copy(BufBytes, OutBytes, BytesRead);
  }
  else
  {
    throw(new ApplicationException("Порт не был открыт"));
  }
```

```
    return OutBytes;
  }

  // Передача данных
  public void Write(byte[] WriteBytes)
  {
    if (hPortHanle!=INVALID_HANDLE_VALUE)
    {
      int BytesWritten = 0;
      WriteFile(hPortHanle,WriteBytes,WriteBytes.Length,
                                  ref BytesWritten, 0);
    }
    else
    {
      throw(new ApplicationException("Порт не был открыт"));
    }
  }
  // Описание констант Win32 API
  private const uint GENERIC_READ = 0x80000000;
  private const uint GENERIC_WRITE = 0x40000000;
  private const int OPEN_EXISTING = 3;
  private const int INVALID_HANDLE_VALUE = -1;

  [StructLayout(LayoutKind.Sequential)]
  public struct DCB
  {
    public int DCBlength;
    public int BaudRate;
    /*
      public int fBinary;
      public int fParity;
      public int fOutxCtsFlow;
      public int fOutxDsrFlow;
      public int fDtrControl;
      public int fDsrSensitivity;
      public int fTXContinueOnXoff;
      public int fOutX;
      public int fInX;
      public int fErrorChar;
      public int fNull;
      public int fRtsControl;
      public int fAbortOnError;
      public int fDummy2;
      */
```

```
  public uint flags;
  public ushort wReserved;
  public ushort XonLim;
  public ushort XoffLim;
  public byte ByteSize;
  public byte Parity;
  public byte StopBits;
  public char XonChar;
  public char XoffChar;
  public char ErrorChar;
  public char EofChar;
  public char EvtChar;
  public ushort wReserved1;
}

[DllImport("kernel32.dll")]
private static extern int CreateFile (
  string lpFileName,
  uint dwDesiredAccess,
  int dwShareMode,
  int lpSecurityAttributes,
  int dwCreationDisposition,
  int dwFlagsAndAttributes,
  int hTemplateFile
  );
[DllImport("kernel32.dll")]
private static extern bool GetCommState (
  int hFile,    // дескриптор файла (порта)
  ref DCB lpDCB // структура DCB
  );
[DllImport("kernel32.dll")]
private static extern bool SetCommState (
  int hFile,    // дескриптор файла (порта)
  ref DCB lpDCB // структура DCB
  );
[DllImport("kernel32.dll")]
private static extern bool ReadFile (
  int hFile,          // дескриптор файла (порта)
  byte[] lpBuffer,    // буфер
  int nNumberOfBytesToRead,     // размер буфера
  ref int lpNumberOfBytesRead,  // реально прочитано
  int lpOverlapped    // 0 для синхронных операций
);
```

```
    [DllImport("kernel32.dll")]
    private static extern bool WriteFile (
      int hFile,            // дескриптор файла (порта)
      byte[] lpBuffer,                // буфер данных
      int nNumberOfBytesToWrite,      // число байтов данных
      ref int lpNumberOfBytesWritten, // реально переданное
                                      // число байтов
      int lpOverlapped      // 0 для синхронных операций
    );
    [DllImport("kernel32.dll")]
    private static extern bool CloseHandle (
      int hObject // дескриптор файла (порта)
    );
  }
}
```

27.8.3. Вариант 3

В C# 2.0 (Visual Studio 2005) имеется специальный компонент `System.IO.Ports.SerialPort`. События `DataReveived`, `ErrorReceived` и `PinChanged` позволяют реализовать всю необходимую функциональность.

ГЛАВА 28

Изображения

28.1. Изменение размеров картинки

Изменение размера производится с помощью метода `DrawImage`, затем полученное изображение сохраняется в выходной файл (листинг 28.1).

Листинг 28.1. Изменение размеров картинки

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
using System.IO;

namespace ImageResize
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Исходный файл
      string fromFileName = @"input.jpg";
      // Файл результата
      string toFileName   = @"output.jpg";

      // Нужные размеры
      int sizeX = 200;
      int sizeY = 200;
```

```
      // Загружаем исходную картинку
      Image image = Image.FromFile(fromFileName);

      // Создаем bitmap нужного размера
      Bitmap bmp = new Bitmap(sizeX, sizeY);
      bmp.MakeTransparent();

      // Рисуем на bitmap-картинку
      Graphics graphics = Graphics.FromImage(bmp);
      graphics.CompositingQuality = CompositingQuality.HighQuality;
      graphics.InterpolationMode =
                    InterpolationMode.HighQualityBicubic;
      graphics.DrawImage(image, 0, 0, sizeX, sizeY);
      graphics.Flush();

      // Сохраняем результат (с сохранением RawFormat)
      File.Delete(toFileName);
      FileStream stream = new FileStream(toFileName,
                                         FileMode.Create);
      bmp.Save(stream, image.RawFormat);
      stream.Close();

      // image больше не нужен
      image.Dispose();
    }
  }
}
```

28.2. Доступ к пикселам изображения

Методы `GetPixel(xpos, ypos)` и `SetPixel(xpos, ypos)` позволяют получить и установить цвет конкретного пиксела изображения.

28.3. Создание негатива изображения

Негатив цвета вычисляется с помощью RGB-составляющих (листинг 28.2).

Листинг 28.2. Создание негатива изображения

```
using System;
using System.Drawing;

namespace ImageNegative
{
  class Class1
```

```
  {
    [STAThread]
    static void Main(string[] args)
    {
      Bitmap bmp = new Bitmap(@"test.bmp");
      for(int x = 0; x < bmp.Width; x++)
      {
        for(int y = 0; y < bmp.Height; y++)
        {
          System.Drawing.Color c = bmp.GetPixel(x,y);
          bmp.SetPixel(x,y, System.Drawing.Color.FromArgb(
                    255 - c.R, 255 - c.G, 255 - c.B));
        }
      }
      bmp.Save("out.bmp");
    }
  }
}
```

28.4. Изменение цветовой матрицы изображения

С помощью матрицы цветов можно изменять цвета изображения, не используя пиксельных преобразований.

28.4.1. Преобразование цветов изображения

Пусть в компоненте `pictureBox1` загружена некоторая картинка (GIF или JPG).

```
private System.Drawing.Imaging.ColorMatrix CreateSepiaMatrix()
{
  // Задаем преобразование
  // New Red = R*.1 + G*.4 + B*.7
  // New Green = R*.2 + G*.5 + B*.8
  // New Blue = R*.3 + G*.6 + B*.9
  return new System.Drawing.Imaging.ColorMatrix(new float[][]
  {
    new float[] {0.1f, 0.4f, 0.7f, 0, 0},
    new float[] {0.2f, 0.5f, 0.8f, 0, 0},
    new float[] {0.3f, 0.6f, 0.9f, 0, 0},
    new float[] { 0, 0, 0, 1, 0},
    new float[] { 0, 0, 0, 0, 1}
  });
}
```

```
private void button1_Click(object sender, System.EventArgs e)
{
  // Получить изображение (jpg или gif)
  Image img = pictureBox1.Image;

  // Создать объект атрибутов изображения
  System.Drawing.Imaging.ImageAttributes imageAttrs =
  new System.Drawing.Imaging.ImageAttributes();
  // Создать атрибуты по матрице преобразования
  imageAttrs.SetColorMatrix(CreateSepiaMatrix());
  // Нарисовать новое изображение с помощью преобразования
  using(Graphics g = Graphics.FromImage(img))
  {
  g.DrawImage(img, new Rectangle(0, 0, img.Width, img.Height),
   0, 0, img.Width, img.Height, GraphicsUnit.Pixel, imageAttrs);
  }
  // Обновить рисунок
  pictureBox1.Invalidate();
}
```

28.4.2. Создание серого изображения

Серое изображение создается с помощью указания новых атрибутов (листинг 28.3).

Листинг 28.3. Создание серого изображения

```
using System;
using System.Drawing;
using System.Drawing.Imaging;

namespace ImageGray
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Загружаем входной файл
      Bitmap bmp = new Bitmap(@"Add_IWshRuntimeLibrary.bmp");
      // Атрибуты серого изображения
      ImageAttributes ia = new ImageAttributes();
      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 серое
      Graphics g = Graphics.FromImage(bmp);
      g.DrawImage(bmp, new Rectangle(0, 0,
                           bmp.Width, bmp.Height), 0, 0,
        bmp.Width, bmp.Height,
        GraphicsUnit.Pixel, ia);
      // Сохраняем выходной файл
      bmp.Save("out.bmp");
    }
  }
}
```

28.4.3. Создание "затемненного" изображения

Для создания "затемненного" (как при выключении Windows) изображения используется матрица вида:

```
cm.Matrix00 = 0;
cm.Matrix11 = 0;
cm.Matrix22 = 0;
cm.Matrix33 = 0.25f;
```

28.5. Рисование блокированной кнопки

Изображение для блокированной (disabled) кнопки можно нарисовать с помощью метода `DrawImageDisabled` класса `System.Windows.Forms.ControlPaint`.

28.6. Создание серого оттенка цвета

С помощью специальных коэффициентов можно преобразовать цвет в серые оттенки (автор Джон Скит (Jon Skeet)):

```
// Исходный цвет
Color original = Color.Red;
```

```
// Уровень серого
double greyLevel = original.R*0.299 + original.G*0.587 +
                   original.B*0.144;
if (greyLevel > 255)
{
  greyLevel = 255;
}
// Цвет серого оттенка
Color grayed = Color.FromArgb((int) greyLevel,
                          (int)greyLevel, (int)greyLevel);
```

28.7. Загрузка изображения из буфера обмена

Загрузка изображения из буфера обмена производится с помощью импортированных Win32-функций (листинг 28.4).

Листинг 28.4. Загрузка изображения из буфера обмена

```
using System;
using System.Runtime.InteropServices;
using System.Reflection;
using System.Drawing;
using System.Drawing.Imaging;

namespace ImageFromClipboard
{
  class Class1
  {

public const uint CF_METAFILEPICT = 3;
public const uint CF_ENHMETAFILE = 14;
[DllImport("user32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
public static extern bool OpenClipboard(IntPtr hWndNewOwner);
[DllImport("user32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
public static extern bool CloseClipboard();
[DllImport("user32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
public static extern IntPtr GetClipboardData(uint format);
[DllImport("user32.dll", CharSet=CharSet.Auto, ExactSpelling=true)]
public static extern bool IsClipboardFormatAvailable(uint format);

    [STAThread]
    static void Main(string[] args)
    {
      if (OpenClipboard(new IntPtr(0)))
```

```
      {
        if (IsClipboardFormatAvailable(CF_ENHMETAFILE))
        {
          IntPtr ptr = GetClipboardData(CF_ENHMETAFILE);
          if (!ptr.Equals(new IntPtr(0)))
          {
            Metafile metafile = new Metafile(ptr,true);
            metafile.Save("out.bmp");
          }
        }
        CloseClipboard();
      }
    }
  }
}
```

28.8. Замена цвета в изображении

С помощью класса `System.Drawing.Imaging.ImageAttributes` можно заменить один или несколько цветов в изображении (листинг 28.5).

Листинг 28.5. Замена цвета в изображении

```
using System;
using System.Drawing;
using System.Drawing.Imaging;

namespace ImageReplaceColor
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Загружаем входной файл
      Bitmap bmp = new Bitmap(@"Add_IWshRuntimeLibrary.bmp");

      // Атрибуты серого изображения
      ImageAttributes ia = new ImageAttributes();
      ColorMap[] clrmap = new ColorMap[1]{ new ColorMap() };
      clrmap[0].OldColor = Color.Black;
      clrmap[0].NewColor = Color.Red;
      ia.SetRemapTable(clrmap);
```

```
      // Рисуем
      Graphics g = Graphics.FromImage(bmp);
      g.DrawImage(bmp, new Rectangle(0, 0,
          bmp.Width, bmp.Height), 0, 0,
          bmp.Width, bmp.Height,
          GraphicsUnit.Pixel, ia);
      // Сохраняем выходной файл
      bmp.Save("out.bmp");
    }
  }
}
```

28.9. Преобразование значков в изображение

Метод `ToBitmap()` позволяет преобразовать значок в изображение:

```
Icon ico = new Icon();
Bitmap bmp = ico.ToBitmap();
```

28.10. Создание графической экранной копии формы

28.10.1. Вариант 1

Создание изображения формы производится с помощью импортированного метода `BitBlt` (листинг 28.6).

Листинг 28.6. Создание изображения формы (`ImageCapture`)

```
[System.Runtime.InteropServices.DllImportAttribute("gdi32.dll")]
private static extern bool BitBlt (
  IntPtr hdcDest,
  int nXDest, int nYDest,
  int nWidth, int nHeight,
  IntPtr hdcSrc,
  int nXSrc, int nYSrc,
  System.Int32 dwRop
);

private void button1_Click(object sender, System.EventArgs e)
{
    Graphics g1 = this.CreateGraphics();
    Image MyImage = new Bitmap(this.ClientRectangle.Width,
                        this.ClientRectangle.Height, g1);
```

```
    Graphics g2 = Graphics.FromImage(MyImage);
    IntPtr dc1 = g1.GetHdc();
    IntPtr dc2 = g2.GetHdc();
    BitBlt(dc2, 0, 0, this.ClientRectangle.Width,
            this.ClientRectangle.Height, dc1, 0, 0, 13369376);
    g1.ReleaseHdc(dc1);
    g2.ReleaseHdc(dc2);
    MyImage.Save("out.bmp", ImageFormat.Bmp);
}
```

28.10.2. Вариант 2

Еще одно решение для создания экранной копии (скриншота). Вообще говоря, это решение (листинг 28.7) можно использовать для получения скриншота любой формы по известному дескриптору (handle).

Листинг 28.7. Создание изображения формы (`FormImage`)

```
[Flags]
private enum DrawingOptions
{
  PRF_CHECKVISIBLE = 0x00000001,
  PRF_NONCLIENT    = 0x00000002,
  PRF_CLIENT       = 0x00000004,
  PRF_ERASEBKGND   = 0x00000008,
  PRF_CHILDREN     = 0x00000010,
  PRF_OWNED        = 0x00000020
}

private const int WM_PRINT       = 0x0317;
private const int WM_PRINTCLIENT = 0x0318;

[DllImport("user32.dll")]
private static extern int SendMessage (IntPtr hWnd, int msg,
      IntPtr dc, DrawingOptions opts);

private void button1_Click(object sender, System.EventArgs e)
{
  using (Bitmap bm = new Bitmap(Width + 2, Height + 2))
  {
    using (Graphics g = Graphics.FromImage(bm))
    {
      IntPtr dc = g.GetHdc();
      try
```

```
      {
        SendMessage(Handle, WM_PRINT, dc,
         DrawingOptions.PRF_CHILDREN |
         DrawingOptions.PRF_CLIENT |
         DrawingOptions.PRF_NONCLIENT);
      }
      finally
      {
       g.ReleaseHdc(dc);
      }
      bm.Save("bm.png");
    }
  }
}
```

28.11. Создание графической копии экрана

Создание скриншота экрана производится с помощью импортированного метода `BitBlt` (листинг 28.8).

Листинг 28.8. Создание скриншота экрана

```
using System;
using System.Runtime.InteropServices;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;

namespace CaptureDesktop
{
  class Class1
  {
    [DllImport("user32.dll")]
    public extern static IntPtr GetDesktopWindow ();

    [System.Runtime.InteropServices.DllImport("user32.dll")]
    public static extern IntPtr GetWindowDC (IntPtr hwnd);

    [System.Runtime.InteropServices.DllImport("gdi32.dll")]
    public static extern UInt64 BitBlt
      (IntPtr hDestDC,
      int x, int y, int nWidth, int nHeight,
      IntPtr hSrcDC,
```

```
      int xSrc, int ySrc,
      System.Int32 dwRop);

    [STAThread]
    static void Main(string[] args)
    {
      Image myImage = new Bitmap(Screen.PrimaryScreen.Bounds.Width,
                                 Screen.PrimaryScreen.Bounds.Height);
      Graphics gr1 = Graphics.FromImage(myImage);

      IntPtr dc1 = gr1.GetHdc();
      IntPtr dc2 = GetWindowDC(GetDesktopWindow());

      BitBlt(dc1, 0, 0, Screen.PrimaryScreen.Bounds.Width,
        Screen.PrimaryScreen.Bounds.Height, dc2, 0, 0, 13369376);

      gr1.ReleaseHdc(dc1);

      myImage.Save("screenshot.jpg", ImageFormat.Jpeg);

    }
  }
}
```

28.12. Изменение размера изображения

Изменить размер можно простым преобразованием или с помощью интерполяции (листинг 28.9). Во втором случае преобразование получается более качественным. Кроме того, можно использовать метод создания пиктограмм *(см. разд. 28.13)*.

Листинг 28.9. Изменение размера изображения

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;

namespace ImageChangeSize
{
  class Class1
  {

    [STAThread]
    static void Main(string[] args)
```

```
    {
      // Входной файл
      Bitmap inBmp = new Bitmap(@"Add_IWshRuntimeLibrary.bmp");

      Bitmap outBmp;

      // Простое преобразование размера
      outBmp = new Bitmap(inBmp, inBmp.Width/2, inBmp.Height/2);
      outBmp.Save("out_2.bmp");

      // Интерполированное преобразование размера
      outBmp = new Bitmap(inBmp.Width/2,
                  inBmp.Height/2, PixelFormat.Format24bppRgb);
      Graphics g = Graphics.FromImage(outBmp);
      g.InterpolationMode = InterpolationMode.HighQualityBicubic;
      g.DrawImage(inBmp, 0, 0, outBmp.Width, outBmp.Height);
      outBmp.Save("out_inter.bmp");
    }
  }
}
```

28.13. Создание пиктограммы

Метод `GetThumbnailImage` позволяет создать пиктограмму заданного размера из изображения (листинг 28.10).

Листинг 28.10. Создание пиктограммы

```
using System;
using System.Drawing;
using System.Drawing.Imaging;

namespace ImageThumbnail
{
  class Class1
  {
    public static bool ThumbnailCallback()
    {
      return false;
    }

    [STAThread]
    static void Main(string[] args)
```

```
    {
      // Входной файл
      Bitmap inBmp = new Bitmap(@"Add_IWshRuntimeLibrary.bmp");

      // Преобразование в пиктограмму заданного размера
      Bitmap outBmp = (Bitmap) inBmp.GetThumbnailImage(75, 75,
        new Image.GetThumbnailImageAbort(ThumbnailCallback),
        IntPtr.Zero);

      // Сохраняем выходной файл
      outBmp.Save("out.bmp");
    }
  }
}
```

28.14. Анимированный GIF

В C# имеются средства для работы с анимированным GIF-изображением (листинг 28.11).

Листинг 28.11. Работа с анимированным GIF

```
using System;
using System.IO;
using System.Drawing;
using System.Drawing.Imaging;

namespace AnimatedGif
{
  class Class1
  {
    [STAThread]
    static void Main(string[] args)
    {
      // Загружаем GIF
      Image img = Image.FromFile(@"o009.gif");

      // Число фреймов в анимированном GIF
      FrameDimension dimension =
                new FrameDimension(img.FrameDimensionsList[0]);
      int frameCount = img.GetFrameCount(dimension);
      Console.WriteLine("Фреймов: {0}", frameCount);
```

```
      // Переписываем GIF в набор BMP
      for (int i=0; i<frameCount; i++)
      {
        img.SelectActiveFrame(dimension, i);
        MemoryStream ms = new MemoryStream();
        img.Save(ms, ImageFormat.Bmp);
        Image outImg = Image.FromStream(ms);
        outImg.Save(string.Format("out{0}.bmp", i));
      }
    }
  }
}
```

28.15. Градиентная заливка

Методы класса `System.Drawing.Drawing2D.LinearGradientBrush` позволяют производить градиентную заливку различными способами:

```
private void panel1_Paint(object sender,
                      System.Windows.Forms.PaintEventArgs e)
{
  Rectangle rect = e.ClipRectangle;
  LinearGradientBrush brush2 =
        new LinearGradientBrush(rect, Color.Blue, Color.White,
  LinearGradientMode.ForwardDiagonal);
  e.Graphics.FillEllipse(brush2, rect);
}
```

28.16. Исключение при установке прозрачного фона

При попытке установить фон (`BackColor`) прозрачного цвета (`Color.Transparent`) возникает исключение. Это происходит из-за того, что объект не поддерживает установку прозрачного фона. Избавиться от проблемы позволит вызов метода `SetStyle`:

```
// Разрешаем использовать прозрачный фон
this.SetStyle(ControlStyles.SupportsTransparentBackColor, true);
// Теперь можно задать цвет BackColor
this.BackColor = Color.Transparent;
this.Invalidate();
```

28.17. Как нарисовать математическую формулу?

Шитал Шах (Shital Shah) предлагает специальный модуль MimeTex.dll, созданный на основе библиотек TeX, который позволяет преобразовывать строковое описание формулы в изображение (**www.codeproject.com/dotnet/Eq2Img.asp**).

ПРИЛОЖЕНИЕ 1

Интернет-ресурсы

- Russian Software Development Network
  www.rsdn.ru
- George Shepherd's Windows Forms FAQ
  http://www.syncfusion.com/FAQ/WinForms/default.asp
- Windows API reference for C#, VB.NET & VB6
  http://custom.programming-in.net/
- Организация "горячего" обмена по DDE между Microsoft Excel и приложением .NET
  http://www.codenet.ru/progr/cpp/dotnet/Hot-DDE/
- ASP.NET Mania — все про .NET Framework
  http://www.aspnetmania.com/default.aspx
- The Code Project — Free Source Code and Tutorials
  http://www.codeproject.com/
- Microsoft Developer Network
  http://msdn2.microsoft.com/en-us/default.aspx
- Configuration FAQ
  http://www.grimes.demon.co.uk/dotnet/configFAQ.htm
- Несколько альтернативных способов отправить e-mail
  http://sonext.biz/node/353
- 15 Seconds: asp tutorials, asp.net tutorials, ASP programming sample code, and Microsoft news from 15 Seconds
  www.15seconds.com.

ПРИЛОЖЕНИЕ 2

Описание компакт-диска

Компакт-диск содержит файлы проектов, относящиеся к соответствующим разделам книги. Проекты размещены в папках в соответствии с главами.

- папка \Chapt 01-12 — проекты к *главам 1—12* (табл. П2.1);
- папка \Chapt 13-14 — проекты к *главам 13—14* (табл. П2.2);
- папка \Chapt 15 — проекты к *главе 15* (табл. П2.3);
- папка \Chapt 16-19 — проекты к *главам 16—19* (табл. П2.4);
- папка \Chapt 20 — проекты к *главе 20* (табл. П2.5);
- папка \Chapt 21 — проекты к *главе 21* (табл. П2.6);
- папка \Chapt 22 — проекты к *главе 22* (табл. П2.7);
- папка \Chapt 23 — проекты к *главе 23* (табл. П2.8);
- папка \Chapt 24 — проекты к *главе 24* (табл. П2.9);
- папка \Chapt 25 — проекты к *главе 25* (табл. П2.10);
- папка \Chapt 26 — проекты к *главе 26* (табл. П2.11);
- папка \Chapt 27 — проекты к *главе 27* (табл. П2.12);
- папка \Chapt 28 — проекты к *главе 28* (табл. П2.13).

Таблица П2.1. *Главы 1—12*

Раздел	Название примера
1.4	MessageBox
2.5	IsValueType
2.6	RussianVariable

Таблица П2.1 *(продолжение)*

Раздел	Название примера
2.12	VirtualNewMethods
2.13	Equals
2.14	OverrideOperation
2.15	ParamConstructor
2.16	DelegateExample
2.17	PropertyTest
2.18	IndexProperty
2.20	Interfaces
2.24	Params
2.25	Pointers
2.26	WeakReference
2.27	Resource
3.6	StaticStructArray
3.9	BitVector32Test
4.1	FindCustomContructor Reflection1
4.2	FindMethod
4.4	Reflection2
4.6	Reflection3
4.7	Reflection4
4.8	LoadWithPartialName
4.10	ResolveAssembly
5.6	TraceCallStack
5.7.3	QueryPerformanceCounter
5.11	UnhandledExceptionTest
6.1	Args
6.2	CommandLine
6.6	ConfigHandler

Таблица П2.1 *(окончание)*

Раздел	Название примера
7.3	IntParse
7.7	Float2String
7.8	DoubleConvert
7.12	Color2Str
7.13	ColorTranslatorTest
7.15	HTMLTextConverter
8.2	EnumToString
8.4	ArrayTest
8.4.5	ArrayCopyTo
8.10	BitArray
8.11.4	Hashtable
8.11.5	CaseInsensitiveHashtable
8.12	SortedList
	Comparer
8.13	Queue
8.14	Stack
8.15	ArrayListAdapter
9.1	CRC32
9.2	Crypt_DES
	Crypt_DES1
	Crypt_Rijndael
9.4	JScriptEvaluateTest
10.6	JoinString
10.9	UnsafeString
10.10	EscapeChars
11.1	ColorText
11.3	ConsoleOutput
11.4	ConsoleTitle

Таблица П2.2. *Главы 13—14*

Раздел	Название примера
13.1	GetLogicalDrives
13.3	Clipboard2
13.6	GetFilesByMask
14.1	ChangeErrorModeTest
14.2	GetFileSystemEntries
14.3	BrowseForFolder
14.5	DeleteFile
14.11	ReadFile1
14.14	FileSystemWatcher
14.15	ShortFileName

Таблица П2.3. *Глава 15*

Раздел	Название примера
15.1	UriParse
15.6	RasEnumConnections
15.7	SendMail_Socket
	SMTP_Send
	SMTP_Test_Aut
15.8	EnumNetworkDrives
15.10	LogonUser
15.12	DownloadFile
15.14	DownloadWebPage
15.16	InternetAvailable
15.17	HopCount
15.18	Ping2
15.19	FTP_File
15.20	ServerEnum

Таблица П2.4. *Главы 16—19*

Раздел	Название примера
16.2	AddShortcutToDesktop
16.4	MyExtShell
16.5	ShellDemo
17.1	Threads_Create
	Threads_Standard
17.3	Threads_Slot
17.4	Threads_Principal
17.6	Threads_Pool
17.9	Threads_Events
	Threads_WaitAny
18.1	CreateNewProcess
18.2.1	OneInstanceApplication1
18.2.2	OneInstanceApplication2
18.3	ProcessesInformation
18.5	ShowAllGUIProcesses
18.6	RunProcess
18.7	ProcessReference
18.8	WinServiceManagement
19.1	EnumPrinterConnections
19.2	SetDefaultPrinter
19.3	Environment
19.5	DecimalSeparator
19.6	ProcessorCount
19.8	EventLogTest
19.9	EventLogDelete
19.10	EventLogList
19.11	Beep
19.12	Random
19.14	GetFrameworkPathApplication
19.15	PlayWav

Таблица П2.4 *(окончание)*

Раздел	Название примера
19.16	ReadRegistry
19.18	SerializeTest
19.19	FileTypeByExt
19.20	GenerateGUID
19.21	LockUp
19.25	SAPI5

Таблица П2.5. *Глава 20*

Раздел	Название примера
20.3	RegexReplace
20.5	ParseCSV
20.6	MacroNames
20.9	RegExTest

Таблица П2.6. *Глава 21*

Раздел	Название примера
21.1—21.16	XML-tests
21.19	XSD

Таблица П2.7. *Глава 22*

Раздел	Название примера
22.1	TreyIcon
22.3	InvisibleForm
22.4	EnumColors
22.6	CustomExceptionHandler
22.7	IMessageFilter
	Win32MessageFilter

Таблица П2.7 *(окончание)*

Раздел	Название примера
22.14	SystemEvents
22.15	WebBrowser1
22.17	DrawByHwnd
22.19	RegionHit
22.20	WinFormConsole
22.21	AutoCompleteText

Таблица П2.8. *Глава 23*

Раздел	Название примера
23.2	OleDb
23.5	SQLServersEnum

Таблица П2.9. *Глава 24*

Раздел	Название примера
24.1—24.19	Example_ASP
24.21	AJAX_Test1
24.22	NoAJAX

Таблица П2.10. *Глава 25*

Раздел	Название примера
25.1	ExcelDataTest
25.2	ExcelReader
25.3	Clipboard1
25.7	OutlookCalendar
25.9	WordSpellChecker
	SpellCheckDemo
25.10	IEHistory

Таблица П2.11. *Глава 26*

Раздел	Название примера
26.2	WMITest
26.3	Win32_VideoController
26.4	Win32_ComputerSystem
26.5	Win32_ComputerSystemProduct
26.6	Win32_OperatingSystem
26.7	Win32_OperatingSystem_1
26.8	Win32_Desktop
26.9	Win32_ComputerSystem_1
26.10	Win32_SystemEnclosure
26.11	Win32_ComputerSystem_2
26.12	Win32_Processor
26.13	Win32_Share
26.14	Win32_Share_1
26.15	Win32_NetworkConnection
26.16.1	Win32_LogicalDisk
26.16.2	Win32_LogicalDisk_1
26.17	Win32_Environment
26.18	Win32_BootConfiguration
26.19	Win32_Process
26.20	Win32_Process_1
26.21	Win32_DiskPartition
26.22	Win32_UserAccount
26.23	Win32_Group

Таблица П2.12. *Глава 27*

Раздел	Название примера
27.1	DeviceEnumerator
27.2	DeviceEnumerator1
27.3	DeviceStatus

Таблица П2.12 *(окончание)*

Раздел	Название примера
27.4	DeviceEject
27.5	DeviceMonitor
27.6	USB.DirectX.HIDDeviceList
27.7	DirectInput-Scancodes
27.8.1	MSCOMMTest
27.8.2	ComPortRead

Таблица П2.13. *Глава 28*

Раздел	Название примера
28.1	ImageResize
28.3	ImageNegative
28.4	ImageAttributes
	ImageGray
28.7	ImageFromClipboard
28.8	ImageReplaceColor
28.10.1	ImageCapture
28.10.2	FormImage
28.11	CaptureDesktop
28.12	ImageChangeSize
28.13	ImageThumbnail
28.14	AnimatedGif
28.15	GradientFill
28.16	TransparentControl

Список литературы

1. Агуров П. В. Практика программирования USB. — СПб.: БХВ-Петербург, 2006.
2. Рихтер Д. Программирование на платформе Microsoft .NET Framework Windows для профессионалов. — М.: Microsoft Press, 2003.

Предметный указатель

E

F

G

I

M

N

P

R

S

T

U

V

W

X

Л

М

О

П

P

C

Т

У

Ф

Х

Ш

Я