Вячеслав Понамарев

Программирование на C++ /C# в Visual Studio .NET 2003

- **Платформы .NET и .NET Framework**
  Основные понятия
- **Среда Visual Studio .NET 2003**
  Рекомендации по настройке и расширению среды разработки
- **Языки C++ и C#**
  Примеры создания программ
- **Web-приложения**
  Создание полноценного Web-магазина компьютерной техники

МАСТЕР_ПРОГРАММ_

Вячеслав Понамарев

Программирование на C++ /C# в Visual Studio .NET 2003

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.068+800.92C++/C#
ББК 32.973.26-018.1
П56

Понамарев В. А.

П56 Программирование на C++/C# в Visual Studio .NET 2003. — СПб.: БХВ-Петербург, 2004. — 352 с.: ил.

ISBN 5-94157-402-9

В книге рассматриваются особенности разработки приложений в среде Visual Studio .NET 2003 с применением языков программирования C++ и C#. Проводится сравнительный анализ алгоритмических языков. Показано удобство создания приложений на языке C#, а также аналогичность этого языка языку Visual Basic .NET. Раскрываются основы работы с данными, изучаются технологии ADO и XML. Для повышения наглядности материала используются схемы, рисунки, таблицы, приводятся также поясняющие программы-примеры. Уделяется внимание самостоятельному созданию больших проектов с последующим ознакомлением с механизмом отладки приложений.

Для программистов

УДК 681.3.068+800.92C++/C#
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Яковлева*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Оформление серии	*Via Design*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.01.04.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 28,38.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-402-9

Содержание

Введение

Новая технология .NET постепенно внедряется в большинство языков программирования, которые разрабатываются не только корпорацией Microsoft. Но основу составляют именно языки, входящие в состав Microsoft Visual Studio .NET. В книге будет рассматриваться программирование на двух основных языках: C# и C++ из новой среды разработки Microsoft Visual Studio .NET 2003. В большинстве случаев упор будет делаться на язык C#, т. к. он является на сегодняшний день наиболее соответствующим технологии .NET.

В процессе работы использовалась среда разработки Microsoft Development Environment 2003 Version 7.1.3088. Все примеры, приведенные в книге, тестировались именно в этой версии.

На кого рассчитана книга

Книга предназначена для всех желающих, стремящихся изучить особенности языка программирования C#, который активно продвигается корпорацией Microsoft. Книга рассчитана на подготовленных программистов, знакомых с языками программирования C или C++.

Я благодарен всем читателям, которые присылают свои рецензии о моих книгах. Присылайте свои отзывы и пожелания по этой и другим книгам, написанным мною, по адресу в Интернете: **ponamarev@mail.ru**.

Структура и особенности книги

Книга состоит из шести частей.

Часть I кратко описывает платформы .NET и .NET Framework. Раскрывается общая модель программирования в среде .NET. Изучается механизм работы с компонентами в среде .NET.

В *части II* описывается механизм настройки среды разработки. Раскрываются принципы расширения среды разработки. Изучается работа с кодом. Описывается работа с файлами ресурсов, а также проектами и решениями.

Часть III рассказывает об языках C++ и C#. Приводится краткое описание синтаксиса двух языков. Рассматриваются примеры небольших программ на языках C++ и C#. Проводится сравнительный анализ языков и описываются основные возможности данных языков программирования.

Часть IV посвящена раскрытию основ работы с наборами данных, технологиями ADO и XML.

Часть V посвящена созданию больших проектов в среде Visual Studio .NET 2003. Рассматриваются принципы создания Web-приложений на языках C++ и C#. На одном большом примере описывается создание Web-магазина. Изучаются механизмы отладки приложений.

В *части VI* (приложениях) вы найдете дополнительную информацию.

Приложение 1 содержит ссылки на сайты, которые посвящены технологии .NET и языку C#.

Приложение 2 объясняет различие между редакциями Visual Studio .NET 2003.

Приложение 3 содержит аппаратные и программные требования, которые предъявляются при установке Visual Studio .NET 2003.

Приложение 4 приводит список языков, которые позволяют компилировать программы в IL-код.

Приложение 5 содержит краткое описание типов данных, используемых в языках .NET.

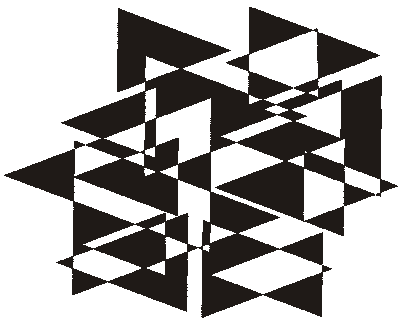
Наконец, в *приложении 6* приводится краткий список сокращений и терминов, которые используются в книгах и справочных материалах, посвященных технологии .NET.

Соглашения, используемые в книге

В книге были использованы следующие издательские соглашения:

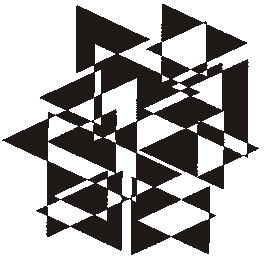
- листинги, идентификаторы, имена функций, классов, переменных, команд, которые вы можете увидеть на экране монитора, выделены моноширинным шрифтом;
- *курсивом* выделены новые термины, которые впервые встречаются в тексте книги. А также таким образом помечены важные места текста;
- **полужирным** шрифтом выделены имена элементов интерфейса: окон, пунктов меню, вкладок и т. д.;
- важные моменты, на которые стоит обратить внимание, выделены в примечания;
- названия клавиш клавиатуры заключены в треугольные скобки, например <F1>. Для иллюстрации одновременного нажатия нескольких клавиш используется символ "+", например <Ctrl>+<Alt>+<O>.

ЧАСТЬ I



Платформа .NET

В этой части раскрываются основы платформ .NET и .NET Framework. Описывается общая модель программирования, языковая интеграция, а также язык CLR. Кроме того, в этой части вы найдете общее описание работы с компонентами .NET.



ГЛАВА 1

Платформы .NET и .NET Framework

В этой главе приводятся сведения о платформах .NET и .NET Framework. Описываются цели создания и преимущества платформ .NET и .NET Framework.

Что такое Microsoft .NET

Первое заявление корпорации Microsoft о переходе на платформу .NET было летом 2000-го года на конференции PDC 2000 в городе Орландо штата Флорида.

Платформа .NET создавалась как среда разработки приложений под Windows с новым интерфейсом программирования. В .NET интегрированы службы компонентов COM+, работа с XML, среда Web-разработки ASP, а также ориентация на создание интернет-приложений и поддержка новейших Web-сервисов (SOAP, UDDI и WSDL).

Вся платформа .NET состоит из различных продуктов, которые можно условно разделить на четыре группы:

- *средства разработки* — т. е. языки программирования (Visual C, C#, Visual Basic .NET, Visual Java), среда Common Language Runtime (CLR, общеязыковая среда выполнения), библиотека классов для создания разнообразных приложений, а также инструментальная среда разработки Visual Studio .NET;
- *Web-сервисы* — т. е. возможность применения коммерческих Web-сервисов (таких, как .NET MyServices), которые необходимы для создания Web-приложений, требующих идентификации пользователей;
- *специализированные серверы* — набор серверов SQL Server, Exchange Server, BizTalk и др., объединенных в одно семейство серверов .NET Enterprise Servers. Эти серверы обеспечивают работу с базами данных, с электронной почтой, и многое другое;

- *поддержка устройств* — встроенная поддержка новых устройств ("умных" устройств), которые могут работать с технологиями .NET (например, мобильные телефоны).

Архитектура платформы .NET

В упрощенном общем виде платформа .NET состоит из пяти основных компонентов (рис. 1.1).

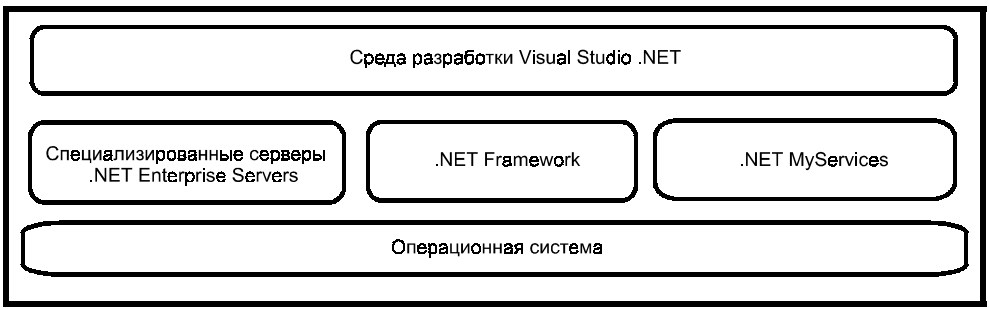


Рис. 1.1. Архитектура платформы .NET

На нижнем уровне платформы находится операционная система (Windows XP, Windows 2000, Windows ME или Windows CE). Кроме перечисленных операционных систем может использоваться и другое специализированное программное обеспечение для так называемых "умных" устройств.

На уровне, расположенном выше уровня операционных систем (рис. 1.1), находятся сразу три компонента:

- *специализированные серверы .NET Enterprise Servers* — набор серверных продуктов, таких как Application Center, BizTalk Server, Commerce Server, Exchange Server, Host Integration Server, Internet Security Acceleration Server и SQL Server;
- *набор Web-сервисов .NET MyServices* — представляющих собой готовые блоки кода, которые разработчик может включать (за дополнительную плату) в свои проекты;
- *.NET Framework* — новая инфраструктура разработки и исполнения Windows-приложений, которая включает в себя общеязыковую среду выполнения CLR, а также общую структуру классов, которые можно использовать в любом языке программирования, относящемся к семейству .NET.

На верхнем уровне архитектуры .NET располагается среда разработки приложений Visual Studio .NET. *Visual Studio .NET* — это интегрированная среда

разработки (Integrated Development Environment, IDE), которая содержит разнообразные средства для создания приложений на языках .NET.

Цели создания .NET Framework

Одной из главных целей создания .NET Framework является упрощение построения Windows-приложений.

Для реализации таких приложений было создано большое количество сред разработки, каждая из которых отличалась от других библиотеками классов, синтаксисом, программным интерфейсом приложений (API, Application Programming Interface) и др. Никакой стандартизации между этими интерфейсами и библиотеками классов просто не существовало.

Использование .NET Framework позволяет программисту использовать набор базовых классов, который подходит для любого языка программирования, относящегося к .NET (C++, C#, Visual Basic .NET и др.). Для успешной работы с любым из этих языков программирования не нужно всякий раз изучать новый API.

Перечислим основные цели создания .NET Framework:

- обеспечение одинаковых возможностей объектно-ориентированного программирования, независимо от используемого языка программирования;
- обеспечение неконфликтности версий создаваемого программного обеспечения и упрощение развертывания приложений, за счет регистраций общих DLL (Dynamic Link Library, динамически подключаемая библиотека) в глобальном кэше сборок (Global Assembly Cache, GAC);
- обеспечение безопасности выполнения кода приложения. .NET Framework предоставляет множество функций безопасности, обеспечивающих надежную защиту от несанкционированного проникновения в приложение или систему. В частности, вы можете защитить даже определенную часть исполняемого кода (например, метод).

Рассмотрим архитектуру .NET Framework.

.NET Framework

В общем виде, .NET Framework является простым системным приложением, работающим в операционной системе, такой как Windows (хотя, могут использоваться и другие операционные системы, например, относящиеся к семейству UNIX).

.NET Framework состоит из двух основных частей:

- *CLR* (Common Language Runtime, общеязыковая среда выполнения);
- *библиотеки классов .NET Framework.*

На рис. 1.2 представлена упрощенная схема архитектуры .NET Framework.

Главным компонентом .NET Framework является CLR. О среде CLR речь пойдет в *гл. 2.* Но пока отметим, что CLR занимается активизацией объектов, отладкой, обработкой исключений, проверкой типов, распределением памяти и ресурсов для объектов, а также осуществляет "сборку мусора", т. е. при прекращении использования того или иного объекта происходит автоматическое уничтожение объекта. Таким образом, CLR осуществляет те же самые действия, которые выполняет виртуальная Java-машина (Java Virtual Machine, JVM) в языке Java.

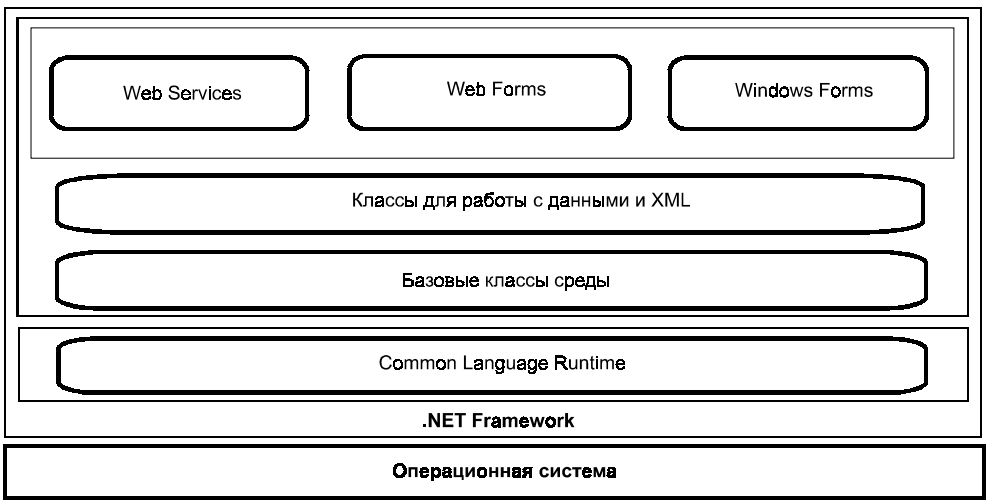


Рис. 1.2. Архитектура .NET Framework

CLR и JVM во многом похожи, но отличаются уже тем, что JVM поддерживает только язык программирования Java, а CLR поддерживает любые языки программирования, которые можно представить на общем для языков .NET промежуточном языке Microsoft Intermediate Language (MSIL, промежуточный язык Microsoft).

Другое отличие JVM и CLR заключается в том, что код Java с помощью JVM может работать на различных платформах (на которых установлена сама JVM), а код .NET работает пока только на платформе Windows.

Библиотеку классов .NET Framework можно условно поделить на несколько частей, согласно их предназначению.

Первая часть — набор базовых классов среды. Это стандартные классы, обеспечивающие работу со строками, ввод и вывод данных, многопоточность, работу с сетью, безопасность и многое другое. Базовые классы похожи по своему составу и функциональности на такие классы, как MFC, ATL и др.

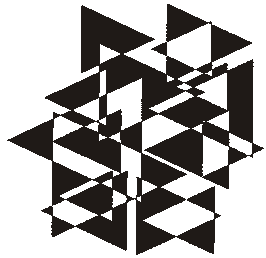
Вторая часть — это набор классов для работы с данными и XML (eXtensible Markup Language). Классы для работы с данными включают в себя класс работы с SQL (Structured Query Language, язык структурированных запросов), а также набор классов ADO.NET для манипулирования постоянными данными. Классы для работы с XML позволяют манипулировать XML-данными, а также выполнять XML-преобразования и XML-поиск.

Третья часть библиотеки расширяет возможности классов для работы с данными и XML и представляет три различных набора классов для поддержки трех технологий:

- *классы Web Services* — поддерживают разработку облегченных распределенных компонентов;
- *классы Web Forms* — позволяют быстро и эффективно создавать Web-приложения, в которых используется графический интерфейс пользователя (Graphical User Interface, GUI). Методы создания интерфейсов пользователя практически ничем не отличаются от создания интерфейсов пользователя на том же Visual Basic. Вы просто помещаете нужные элементы управления на Web-форму, задаете значения нужных свойств и создаете обработчики событий;
- *классы Windows Forms* — позволяют создавать обычные Windows-приложения, в которых используются стандартные элементы управления Windows. Вы можете рассматривать этот набор классов как улучшенную версию классов MFC.

Итак, мы кратко рассмотрели, из каких компонентов состоит .NET и .NET Framework. В *гл. 2* речь пойдет о такой важной части .NET Framework, как среда Common Language Runtime. Мы уточним, как в этой среде поддерживаются и исполняются .NET-компоненты, которые носят название *сборок* (assemblies).

ГЛАВА 2



Среда CLR

В этой главе речь пойдет о среде Common Language Runtime. Мы рассмотрим, что такое CLR и исполняемые файлы CLR. Кроме того, раскрывается понятие метаданных, а также сборок и манифестов.

Что такое Common Language Runtime

Общеязыковая среда выполнения (CLR), как уже упоминалось в *гл. 1*, входит в состав .NET Framework. CLR лежит в основе технологии .NET. Она создана, что называется, "с нуля". В основе CLR не лежит ни одна из ранее используемых в различных языках программирования библиотек, таких как MFC, ATL и др.

Среда CLR управляет выполнением кода, написанного на любом из языков семейства .NET. Данная среда занимается созданием различных объектов, выделяет под них системные ресурсы, выполняет проверку безопасности и исполняет код объектов. Кроме того, CLR отвечает за так называемую "сборку мусора", т. е. уничтожение неиспользуемых объектов и освобождению от них системных ресурсов.

Среда CLR оперирует так называемыми *сборками*.

Сборки (assemblies) — представляют собой .NET-компоненты, которые являются переносимыми исполняемыми файлами (Portable Executable, PE). Эти файлы являются файлами с расширениями exe или dll и состоят из метаданных и кода. О метаданных речь пойдет далее в этой главе, а пока рассмотрим исполняемые файлы CLR.

Исполняемые файлы CLR

Рассмотрим на небольшом примере (консольной программе, выводящей текстовое сообщение) особенности языков программирования C++, C# и Visual Basic .NET.

Пример на C++

Язык C++ в среде .NET несколько отличается от раннего C++ тем, что он расширился за счет нескольких специфичных ключевых слов. Это позволяет программам, написанным на C++ .NET, использовать новые возможности (такие, как "сборка мусора").

В листинге 2.1 приведен код простого консольного приложения, выводящего на экран строку текста. Заметьте, что данный код создается средой Visual Studio .NET автоматически, при создании нового консольного проекта при помощи диалогового окна **New Project** (рис. 2.1).

Листинг 2.1. Простое консольное приложение на C++

```
  //  это главный файл проекта для приложения VC++,
  //  сгенерированный мастером приложений (Application Wizard)
#include "stdafx.h"
#using <mscorlib.dll>
using namespace System;
int _tmain()
{
   // TODO: замените код примера, расположенный ниже, на свой собственный
  Console::WriteLine(S"Hello World");
  return 0;
}
```

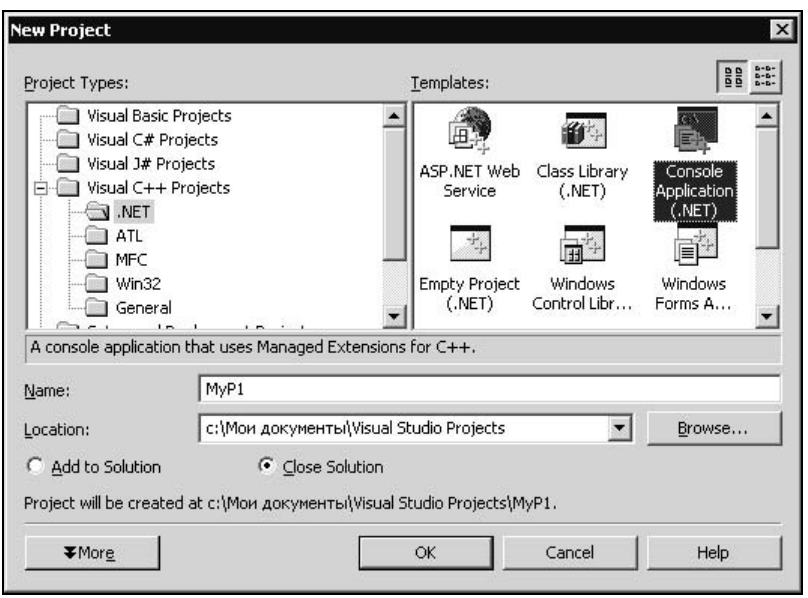


Рис. 2.1. Создание нового консольного приложения на C++

Попробуем разобраться с этим кодом. Как вы можете видеть, это вполне обычная программа на C++. Отличие заключается лишь в использовании директивы компилятора `#using` во второй строке кода. Данная директива делает доступными все типы из указанной библиотеки DLL. Директива `#using` может показаться аналогом директивы `#include`, но в отличие от нее импортирует типы любой сборки .NET, независимо от языка, на котором была написана эта сборка (естественно, это должен быть язык, относящийся к семейству .NET).

Далее все достаточно просто. В методе `_tmain` происходит вызов статического метода `WriteLine()` класса `Console`. Класс `Console` — это тип, который находится в библиотеке mscorlib.dll.

Кроме того, в следующей строке кода, с помощью оператора `using namespace`, компилятору сообщается, что будут использованы типы из пространства имен `System`. В противном случае код строки

```
Console::WriteLine(S"Hello World");
```

пришлось бы заменить на такое выражение:

```
System::Console::WriteLine(S"Hello World");
```

Примечание

Краткие описания пространств имен вы можете прочитать в *гл. 3*.

Если запустить это приложение (клавиша <F5> или пункт меню **Debug | Start**), то вы увидите результат (рис. 2.2).

Рис. 2.2. Результат выполнения консольного приложения

Пример на Visual Basic .NET

В листинге 2.2 приведен пример консольного приложения, выводящего одну строку текста на экран, написанного на языке Visual Basic .NET.

Листинг 2.2. Простое консольное приложение на Visual Basic .NET

```
Module Module1
Sub Main()
Console.WriteLine("Hello World")
End Sub
End Module
```

Как вы можете видеть, вызывается тот же самый метод `WriteLine()` класса `Console` (см. листинг 2.1). Если внимательно посмотреть на информацию о данной сборке — файл AssemblyInfo.vb, то будет виден следующий текст (листинг 2.3).

Листинг 2.3. Содержимое файла Assembly.vb

```
Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices

' общая информация о сборке устанавливается с помощью следующего набора
' атрибутов. Вы можете изменить значения этих атрибутов для задания
' информации о сборке

' просмотр значений атрибутов сборки

<Assembly: AssemblyTitle("")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("")>
<Assembly: AssemblyCopyright("")>
<Assembly: AssemblyTrademark("")>
<Assembly: CLSCompliant(True)>

' следующий GUID является идентификатором ID библиотеки типов, если
' данный проект запланирован как COM-проект
<Assembly: Guid("B8980C3E-3F83-4174-A9CA-C4114ACD51F3")>

' информация о версии данной сборки состоит из следующих четырех значений
'
'   Major Version
'   Minor Version
```

```
'   Build Number
'   Revision
'
' вы можете указать все значения самостоятельно, или использовать
' значения по умолчанию, с помощью знака '*', как показано ниже

<Assembly: AssemblyVersion("1.0.*")>
```

Как видно из листинга 2.3, в самом начале программы происходит импорт типов из пространств имен `System`, `System.Reflection` и `System.Runtime.InteropServices`. Таким образом, используется то же самое пространство имен `System`, как и в примере на C++ (см. листинг 2.1).

Пример на C#

Рассмотрим реализацию консольной программы вывода текстового сообщения на языке C#, который был разработан фирмой Microsoft специально для среды .NET. Язык C# заимствовал синтаксис у языков C++ и Java. Это очень простой для понимания язык. Кстати, большинство утилит и классов .NET написаны именно на этом языке. Итак, пример простого консольного приложения на C# приведен в листинге 2.4.

Листинг 2.4. Простое консольное приложение на C#

```
using System;

namespace ConsoleApplication3
{
   ///  <summary>
   ///  общее описание класса Class1
   ///  </summary>
   class Class1
   {
     ///  <summary>
     ///  главная точка входа в приложение
     ///  </summary>
     [STAThread]
     static void Main(string[] args)
     {
        //
        //  TODO: стартовый код приложения добавьте здесь
        //
        Console.WriteLine("Hello World");
     }
   }
}
```

Здесь также используется пространство имен `System`. Отличие программы на C# от C++ заключается фактически лишь в том, что метод `Main()` в программе на C# является открытой статической функцией класса (в нашем примере — класса `Class1`), а не глобальной функцией, как в C++.

Если внимательно посмотреть на тексты листингов 2.2, 2.3 и 2.4, то можно заметить аналогию между языками Visual Basic .NET и C#. В Visual Basic .NET используются ключевые слова `Import` и `Module`, а в C# — `using` и `class`. Ну и конечно, в C# метод обозначается ключевым словом `void`, а в Visual Basic — `Sub`.

Переносимые исполняемые файлы .NET

Любой исполняемый файл операционной системы Windows (с расширениями exe или dll) должен соответствовать специальному формату *PE* (Portable Executable, переносимый исполняемый файл).

Формат PE является производным от формата *COFF* (Common Object File Format, общий объектный формат файлов).

Операционная система Windows "понимает" формат PE-файлов, поэтому может загружать и исполнять файлы с расширениями exe и dll. Таким образом, любой компилятор генерирует исполняемые файлы Windows в соответствии с форматами PE или COFF.

Обычные PE-файлы Windows можно разделить на две секции:

- *заголовки PE/COFF* (включая ссылки на содержимое PE-файла);
- *двоичные образы*. Включают в себя несколько секций:
  - .Data;
  - .Rdata;
  - .Rsrc;
  - .Text;
  - другие секции, созданные с помощью компилятора языка C++, используя директиву компилятора `pragma`. В этих секциях можно хранить, например, зашифрованные данные.

Рассмотрим внутреннее строение обычного PE-файла Windows (рис. 2.3).

Для поддержки новых возможностей среды CLR Microsoft немного изменила формат PE-файла, генерируемого средой .NET. А именно, были добавлены несколько новых секций (рис. 2.4).

Добавлены следующие секции:

- *заголовок CLR* — содержит информацию, указывающую, что PE-файл является исполняемым файлом .NET;

Рис. 2.3. Формат обычного PE-файла Windows

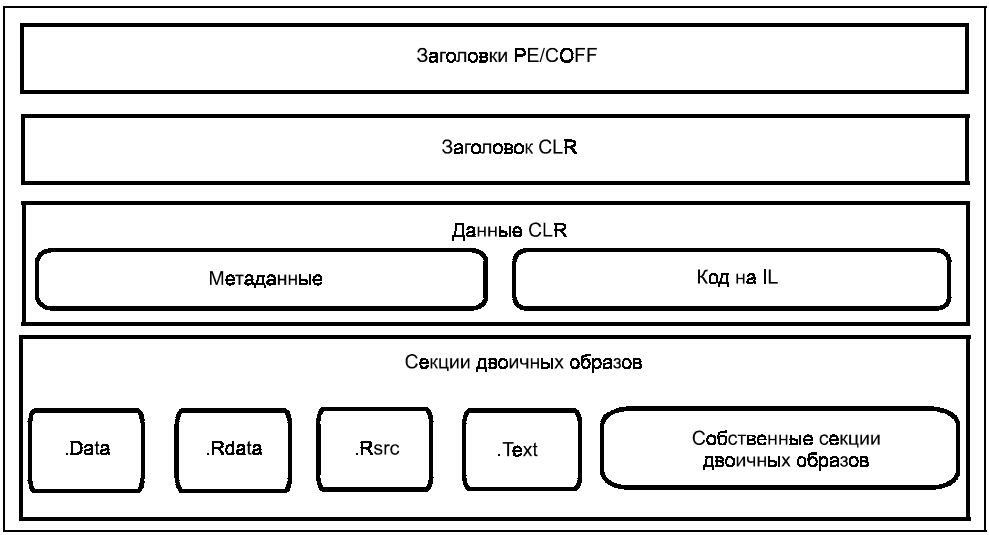


Рис. 2.4. Формат PE-файла, генерируемый средой .NET

- *данные CLR* — определяют, как будет выполняться программа. Состоят из:
  - *метаданных*;
  - *IL-кода*.

Рассмотрим теперь непосредственно метаданные и IL-код.

Метаданные

Метаданные (metadata) — это данные о данных. Другими словами, метаданные — это данные о ресурсах (например, к метаданным относятся определения типов, сведения о версиях, ссылки на внешние сборки и многое другое).

В среде .NET метаданные представляют собой механизм, применяемый всеми компиляторами .NET-языков и утилитами .NET. Таким образом, метаданные служат для описания всех типов, которые используются данной сборкой .NET. Метаданные детально описывают всю сборку (идентификатор сборки, ссылочные и экспортируемые типы, а также требования безопасности для исполнения сборки).

Ранее похожую информацию могли предоставлять библиотеки типов (type library) в технологии COM (Component Object Model, объектная модель компонентов). Но метаданные предоставляют больше информации. Они включают в себя еще многое другое (описание сборки и модулей, описание классов, методов, свойств, полей, событий, интерфейсов и др.).

Просмотр метаданных

Для просмотра метаданных и IL-кода PE-файла, созданного с помощью компилятора .NET, вы можете использовать специальную утилиту — IL-дизассемблер (ildasm.exe), который поставляется вместе с Visual Studio .NET 2003. Данная утилита по умолчанию располагается в папке, в которую вы установили Visual Studio .NET 2003, по следующему пути: **| SDK | v1.1 | Bin**. Например:

C: | Program Files | Microsoft Visual Studio .NET 2003 | SDK | v1.1 | Bin.

Примечание

Если у вас установлена более ранняя версия Visual Studio .NET, то путь к утилите ildasm.exe может несколько отличаться, например, так: **C: | Program Files | Microsoft Visual Studio .NET | FrameworkSDK | Bin**.

Запустим данную утилиту и откроем с ее помощью ранее созданного нами файла Hello.exe (рис. 2.5).

Как вы можете видеть, утилита ildasm.exe отображает метаданные PE-файла в иерархическом виде.

Щелкнем дважды левой кнопкой мыши на узле **MANIFEST**. Откроется дополнительное окно, в котором вы сможете увидеть такой текст (здесь он сокращен для экономии места):

```
.assembly extern mscorlib
{
...
}
.assembly extern Microsoft.VisualC
{
...
}
```

```
.assembly Hello
{
...
}
.module Hello.exe
  // MVID: {D69FDF78-C2FF-4884-B6C9-641AA3DC687F}
...
```

Первые две инструкции языка IL `.assembly extern` означают, что данный PE-файл ссылается на две внешние сборки: `mscorlib` и `Microsoft.VisualC`. Следующая инструкция `.assembly` описывает данную сборку `Hello`. Информация из блоков `.assembly` носит название манифестов.

Наконец, сразу после манифестов вы можете видеть инструкцию IL `.module`, которая сообщает имя модуля (Hello.exe).

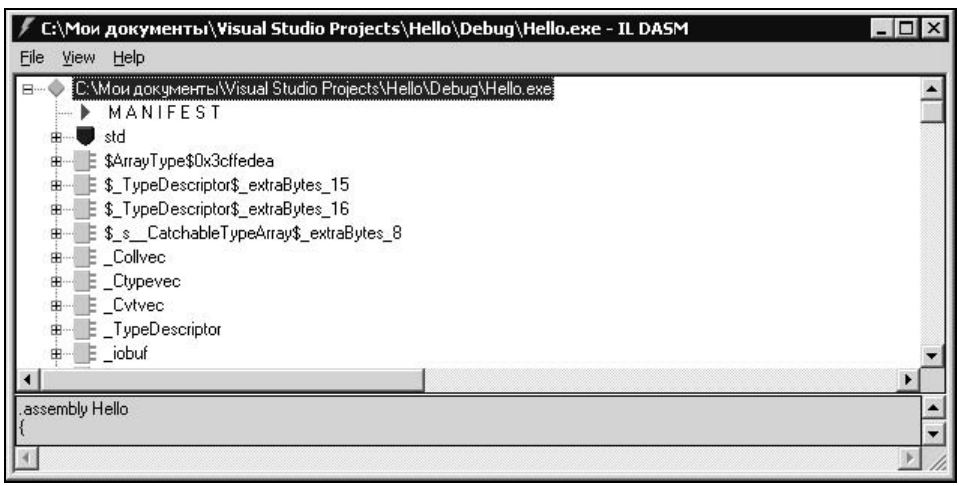


Рис. 2.5. Содержимое файла Hello.exe в утилите ildasm.exe

Рассмотрим еще один узел в иерархии нашего PE-файла, а именно:

main: int32 modopt([mscorlib]System.Runtime.CompilerServices.CallConvCdecl)()

Щелкнув дважды левой кнопкой мыши на этом узле, вы можете видеть в открывшемся окне следующий текст:

```
.method public static int32
modopt([mscorlib]System.Runtime.CompilerServices.CallConvCdecl)
main() cil managed
{
   .vtentry 1 : 1
```

```
    //  code size    14 (0xe)
   .maxstack 1
   IL_0000: ldstr "Hello World"
   IL_0005: call    void [mscorlib]System.Console::WriteLine(string)
   IL_000a: ldc.i4.0
   IL_000b: br.s    IL_000d
   IL_000d: ret
}   //  end of method 'Global Functions'::main
```

Как вы уже догадались, это непосредственно IL-код нашего метода `main()`, выводящего на экран строку текста.

Если вы хотите получить полную информацию о PE-файле в одном текстовом файле, то можете использовать комбинацию клавиш <Ctrl>+<D> в окне дизассемблера ildasm.

Таким образом, любой PE-файл, откомпилированный с помощью языков .NET, можно успешно просмотреть посредством данной утилиты.

Сборки и манифесты

Как уже упоминалось ранее, *сборки* представляют собой файл с расширениями exe или dll, а манифест — это описание или метаданные сборки (версия, имя, используемые другие сборки и т. д.).

Прежде термин "компонент" использовался для обозначения как классов COM, так и для COM-модулей (представляющих собой файлы с расширениями exe и dll). С появлением технологии .NET понятие сборки заменило собой понятие COM-модуля.

Для уникальности типов ранее (в технологии COM) использовались *глобальные уникальные идентификаторы* (GUID). Но таких идентификаторов было огромное количество, это идентификаторы приложений, библиотек, классов, интерфейсов и т. д. Все это затрудняло разработку приложений, т. к. в коде постоянно приходилось использовать числовые значения этих идентификаторов.

В технологии .NET ссылки на любой тип основываются на его имени и пространстве имен. Кроме того, для обеспечения уникальности используются единственные пары открытого и закрытого ключа (как в криптографии с открытым ключом).

Так как манифест сборки содержит информацию о ссылках на другие сборки, теперь отпадает необходимость хранения в реестре указаний по активизации компонентов и маршаллингу (как при использовании технологии COM). Загрузка необходимых типов из внешних сборок будет производиться, основываясь на манифесте сборки, тогда, когда эти типы понадобятся

приложению. Таким образом, можно уменьшить размер загружаемого приложения за счет применения небольших внешних сборок, на которые будут указаны ссылки в манифесте. Необходимые типы будут подгружаться из внешних сборок по мере надобности автоматически без перезагрузки системы или регистрации этих типов.

Сборки вообще являются одним из самых важных элементов технологии .NET. Сборки инкапсулируют в себе все типы, которые были в ней объявлены. Например, имеются две различные сборки: `Library` и `Shop`, в которых определен один и тот же тип `Book`. Для того чтобы обратиться к этому типу, необходимо указать сборку, из которой вы хотите получить данный тип. В противном случае среда CLR не будет знать, к какой сборке относится данный тип, и приложение просто не выполнится.

Рассмотрим теперь (как я и обещал ранее) манифест PE-файла Hello.exe, который мы уже немного изучали в этой главе, более подробно:

```
.assembly extern mscorlib
{
   .publickeytoken = (B7 7A 5C 56 19 34 E0 89)
   .ver 1:0:5000:0
}
.assembly extern Microsoft.VisualC
{
   .publickeytoken = (B0 3F 5F 7F 11 D5 0A 3A)
   .ver 7:0:5000:0
}
.assembly Hello
{
   ...
   .hash algorithm 0x00008004
   .ver 1:0:1325:27733
}
.module Hello.exe
  //  MVID: {D69FDF78-C2FF-4884-B6C9-641AA3DC687F}
```

Как уже было отмечено ранее, наша сборка ссылается на две внешние сборки: `mscorlib` и `Microsoft.VisualC`. Первая сборка необходима для всех приложений, и она будет присутствовать в манифестах любых сборок, которые вы будете создавать.

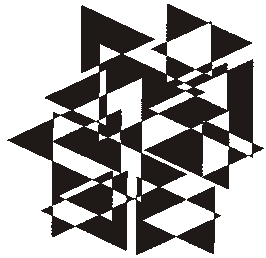
Обратите внимание на тот факт, что в каждой ссылке на эти две сборки имеется значение под названием `.publickeytoken`. Это значение генерируется компилятором, вычисляя хэш-функцию открытого ключа, связанного с каждой из этих сборок. Кроме того, после ключевого слова `.ver` имеется информация о версиях данных сборок.

Далее идет манифест для нашей сборки `Hello`. После ключевых слов `.hash algorithm` указан хэш-алгоритм, который применяется для генерации закрытого ключа. Кроме того, указан номер версии сборки.

Наконец, после ключевого слова `.module` указано имя файла для данной сборки. Заметим, что для данного модуля был сгенерирован GUID, который изменится, если вы перекомпилируете сборку. Таким образом, сравнивая значения GUID, можно утверждать, идентичны или нет два модуля с одинаковым именем.

На этом закончим рассмотрение среды CLR. Хотя это достаточно обширная тема для изучения, но она выходит за рамки нашей книги. В *гл. 3* будет рассказано об общей модели программирования в среде .NET.

ГЛАВА 3



Программирование в среде .NET

В этой главе мы рассмотрим общую модель программирования для среды .NET, а также небольшие примеры программ на базовых языках C++ и C#. Кратко рассмотрим пространства имен .NET Framework.

Общая модель программирования

Среда .NET предоставила программистам общие классы для всех языков программирования .NET. Ранее такое было невозможно. Для создания простого Windows-приложения разработчику предстояло сделать выбор между различными библиотеками (MFC, ATL, Win32 API и др.). Более того, знание классов, функций и интерфейсов, а также прочих возможностей одной библиотеки не могло помочь при использовании другой. Например, знание MFC не позволяло создавать программы на языке Visual Basic, использующем свою библиотеку VB.

В среде .NET достаточно знать, как реализуется та или иная функция на одном языке программирования, чтобы перенести этот код на требуемый язык программирования. Это стало возможным благодаря тому, что все классы, методы и все остальное в среде .NET имеют согласованное представление во всех языках программирования .NET.

Мы уже это видели, когда в *гл. 2* выводили на экран строку текста с помощью одного и того же метода `WriteLine()`, принадлежащего объекту `Console`.

Преимущества такого подхода очевидны. Вам практически не нужно изучать ничего нового при переходе с одного языка программирования платформы .NET на другой, естественно, за исключением синтаксиса самого языка программирования.

Все классы, которые вы можете использовать в любом языке программирования .NET, находятся в различных пространствах имен .NET Framework.