ПРИЛОЖЕНИЕ Б

Независимая от платформы разработка приложений .NET с помощью Mono

Здесь вы ознакомитесь с межплатформенной разработкой приложений на C# и .NET с применением реализации .NET с открытым кодом, которая называется *Mono* (если вам интересно, то "mono" в переводе с испанского — "обезьяна"; создателем Mono была корпорация Ximian, которая использовала для логотипов платформы разнообразные фигурки обезьян). Вы узнаете о роли общеязыковой инфраструктуры, общей области применения Mono и различных инструментах разработки Mono. После освоения предоставленного материала вы сможете при желании самостоятельно углубить свои навыки в разработке приложений Mono.

На заметку! Подробные объяснения межплатформенной разработки в .NET приведены в книге Марка Истона и Джейсона Кинга *Cross-Platform .NET Development: Using Mono, Portable .NET, and Microsoft .NET* (Apress, 2004 г.).

Независимая от платформы природа .NET

Когда-то программистам, использующим язык разработки Microsoft (например, VB6) или среду программирования Microsoft (такую как MFC, COM или ATL), приходилось ограничиваться построением программного обеспечения, которое (в общем) работало только под управлением семейства операционных систем Windows. Многие разработчики .NET, привыкшие к такой привязке к Microsoft, зачастую удивляются, когда узнают, что .NET *не зависит от платформы*. Тем не менее, это так. Создавать, компилировать и выполнять сборки .NET можно под управлением операционных систем, отличающихся от Microsoft.

За счет применения реализаций .NET с открытым кодом, таких как Mono, приложения .NET могут функционировать в средах многих операционных систем, включая Mac OS X, Solaris, AIX и многочисленные разновидности Unix/Linux. Кроме того, Mono предоставляет установочный пакет для Microsoft Windows. Таким

образом, построение и запуск сборок .NET под управлением Windows становятся возможными даже без установки Microsoft .NET Framework SDK или IDE-среды Visual Studio.

На заметку! Имейте в виду, что лучшими средствами создания приложений .NET для семейства операционных систем Windows являются текущие версии Microsoft .NET Framework SDK и Visual Studio.

Даже после того, как разработчики узнали о межплатформенных возможностях кода .NET, они часто полагают, что область независимой от платформы разработки приложений .NET ограничена консольными приложениями уровня "Hello world". Однако в действительности можно строить сборки производственного уровня, которые задействуют ADO.NET, Windows Forms (в дополнение к альтернативным инструментальным наборам, таким как GTK# и Cocoa#), ASP.NET, LINQ и веб-службы XML за счет использования основных пространств имен и языковых средств.

Роль CLI

Межплатформенные возможности .NET реализованы иначе, чем подход, принятый Sun Microsystems для платформы программирования на языке Java. В отличие от Java компания Microsoft не предлагает установщики .NET для Mac, Linux и т.д. Взамен в Microsoft выпустили набор формализованных спецификаций, которые другие субъекты могут применять в качестве дорожной карты при построении дистрибутивов .NET для своих платформ. Все вместе эти спецификации называются *общеязыковой инфраструктурой* (Common Language Infrastructure — CLI).

Как вам должно быть известно, при выпуске языка C# и платформы .NET компания Microsoft отправила две формальные спецификации в Ассоциацию европейских производителей компьютеров (European Computer Manufacturers Association — ECMA). После утверждения эти же спецификации были переданы в Международную организацию по стандартизации (International Organization for Standardization — ISO), где они вскоре были утверждены.

Что нас здесь должно интересовать? Упомянутые две спецификации предоставляют дорожную карту для компаний, разработчиков, университетов и других организаций, которые хотят строить собственные дистрибутивы языка программирования C# и платформы .NET. Вот спецификации, о которых идет речь:

- *ECMA-334* — определяет синтаксис и семантику языка программирования C#;
- *ECMA*-335 — определяет многие детали платформы .NET, коллективно называемые общеязыковой инфраструктурой (CLI).

Спецификация ECMA-334 определяет словарную грамматику C# в исключительно строгой научной манере (как вы могли догадаться, такой уровень детализации очень важен для тех, кто желает реализовывать компилятор языка C#). Среди двух спецификаций ECMA-335 имеет больший объем, поэтому она разбита на шесть разделов (табл. Б.1).

Мы не будем углубляться в детали спецификаций ECMA-334 и ECMA-335 — для понимания процесса построения независимых от платформы сборок .NET знание всех тонкостей, изложенных в этих документах, не требуется.

Таблица Б.1. Разделы спецификации ECMA-335

Раздел	Описание
I. Концепции и архитектура	Описывает общую архитектуру CLI, включая правила системы общих типов, спецификацию общего языка и работу механизма времени выполнения .NET
II. Определение и семантика метаданных	Описывает детали формата метаданных .NET
III. Набор инструкций CIL	Описывает синтаксис и семантику общего промежуточного языка программирования (CIL)
IV. Профили и библиотеки	Предоставляет высокоуровневый обзор минимального и полного набора библиотек классов, которые должны поддерживаться дистрибутивом .NET, совместимым с CLI
V. Форматы обмена отладочной информацией	Описывает переносимый формат обмена отладочной информацией (CILDB). Переносимые файлы CILDB обеспечивают стандартный способ обмена отладочной информацией между производителями и потребителями CLI
VI. Дополнения	Предоставляет собрание разнообразных статей, которые проясняют такие темы, как рекомендации по созданию библиотек классов и детали реализации компилятора CIL

Тем не менее, если вам интересно, то обе спецификации доступны для загрузки на веб-сайте ECMA по адресу `http://www.ecma-international.org/publications/standards/Standard.htm`.

Главные дистрибутивы CLI

В настоящее время кроме среды CLR, разработанной Microsoft, существуют две главные реализации CLI — Microsoft Silverlight и Microsoft NET Compact Framework (табл. Б.2).

Таблица Б.2. Главные дистрибутивы .NET CLI

Дистрибутив CLI	Веб-сайт поддержки	Описание
Mono	`www.mono-project.com`	Mono представляет собой дистрибутив .NET с открытым кодом и коммерческой поддержкой, спонсируемый корпорацией Novell. Предназначен для работы под управлением множества популярных разновидностей Unix/Linux, Mac OS X, Solaris и Windows
Portable .NET	`www.dotgnu.org`	Распространение Portable .NET регламентируется лицензией GNU General Public License. Предназначен для работы под управлением многих операционных систем и архитектур, включая экзотические платформы вроде BeOS, Microsoft Xbox и Sony PlayStation

Каждая из перечисленных в табл. Б.2 реализаций CLI предоставляет полнофункциональный компилятор C#, многочисленные инструменты разработки командной строки, реализацию глобального кеша сборок (GAC), примеры кода, полезную документацию и десятки сборок, которые образуют библиотеки базовых классов.

Помимо реализаций основных библиотек, определенных в разделе IV спецификации ECMA-335, в Mono и Portable .NET предлагаются совместимые с Microsoft реализации библиотек `mscorlib.dll`, `System.Core.dll`, `System.Data.dll`, `System.Web.dll`, `System.Drawing.dll` и `System.Windows.Forms.dll` (а также многих других).

Дистрибутивы Mono и Portable .NET поставляются также с несколькими сборками, которые специально ориентированы на операционные системы Unix/Linux и Mac OS X. Например, Cocoa# является оболочкой .NET для инструментального набора Cocoa, широко используемого при разработке графических пользовательских интерфейсов для Mac OX. Мы не будем исследовать сборки, специфичные для операционных систем, а сосредоточим внимание на технологиях программирования, не зависящих от операционной системы.

На заметку! В этом приложении не рассматривается дистрибутив Portable .NET; однако важно знать, что Mono — не единственный независимый от платформы дистрибутив .NET, доступный в настоящее время. В дополнение к Mono рекомендуется самостоятельно поэкспериментировать с Portable .NET.

Область действия Mono

Учитывая, что Mono является API-интерфейсом, построенным на основе существующих спецификаций ECMA, которые берут начало в Microsoft, вполне корректно предположить, что с выходом новых версий платформы Microsoft .NET дистрибутив Mono постоянно обновляется. Эта технология позволяет создавать веб-сайты ASP.NET, приложения Windows Forms, приложения, взаимодействующие с базами данных посредством ADO.NET, и (конечно же) простые консольные приложения.

Кроме постоянной гонки за основными возможностями различных API-интерфейсов .NET и языка C# технология Mono также предоставляет дистрибутив с открытым кодом Silverlight API под названием *Moonlight*. Он позволяет браузерам, работающим под управлением операционных систем Linux, обслуживать веб-приложения Silverlight/Moonlight. Возможно, вам известно, что Microsoft Silverlight уже включает поддержку Mac OS X, и при наличии Moonlight API эта технология действительно стала межплатформенной.

В Mono также поддерживаются некоторые технологии на основе .NET, у которых нет прямых эквивалентов от Microsoft. К примеру, Mono поставляется с GTK# — оболочкой .NET вокруг популярной инфраструктуры для построения графических пользовательских интерфейсов в Linux, которая называется GTK. Интересным API-интерфейсом на основе Mono является MonoTouch, позволяющий создавать приложения для устройств Apple iPhone и iTouch на языке программирования C#.

Последний интересный момент, касающийся возможностей Mono: подобно .NET Framework SDK от Microsoft, комплект Mono SDK также поддерживает несколько языков программирования для .NET. Помимо C# в Mono предоставляется поддержка компилятора Visual Basic, а также многих других языков программирования, ориентированных на .NET.

Получение и установка Mono

Теперь, когда вы имеете лучшее представление о том, что можно делать с помощью платформы Mono, давайте посмотрим, как получить и установить Mono. Зайдите на веб-сайт Mono (`www.mono-project.com`) и перейдите на вкладку Download (Загрузка). На ней можно загрузить различные установщики (рис. Б.1).

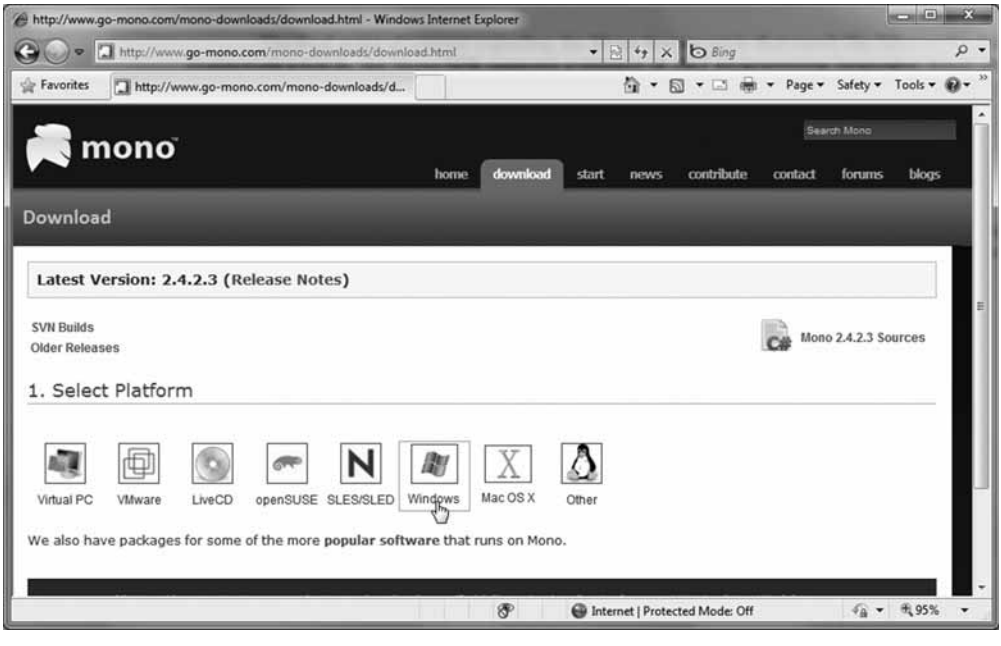

Рис. Б.1. Страница загрузки Mono

Здесь будет описан процесс установки дистрибутива Mono для Windows (эта установка не станет помехой существующей установке Microsoft .NET или Visual Studio). Загрузите текущий стабильный установочный пакет Mono для Microsoft Windows и сохраните программу установки на локальном жестком диске.

После запуска программы установки вам предоставляется возможность установить разнообразные инструменты разработки Mono в дополнение к стандартным библиотекам базовых классов и инструментам программирования на C#. В частности, программа установки запросит о необходимости установки GTK# (API-интерфейс .NET для построения графических пользовательских интерфейсов с открытым кодом, основанный на инструментальном наборе GTK для Linux) и XSP (автономный веб-сервер, подобный веб-серверу разработки ASP.NET от Microsoft). Мы будем предполагать, что выбрана полная установка, включающая все пункты в сценарии установки (рис. Б.2).

Структура каталогов Mono

По умолчанию Mono устанавливается в каталог `C:\Program Files\Mono-<`*версия*`>`, внутри которого находится несколько подкаталогов (рис. Б.3).

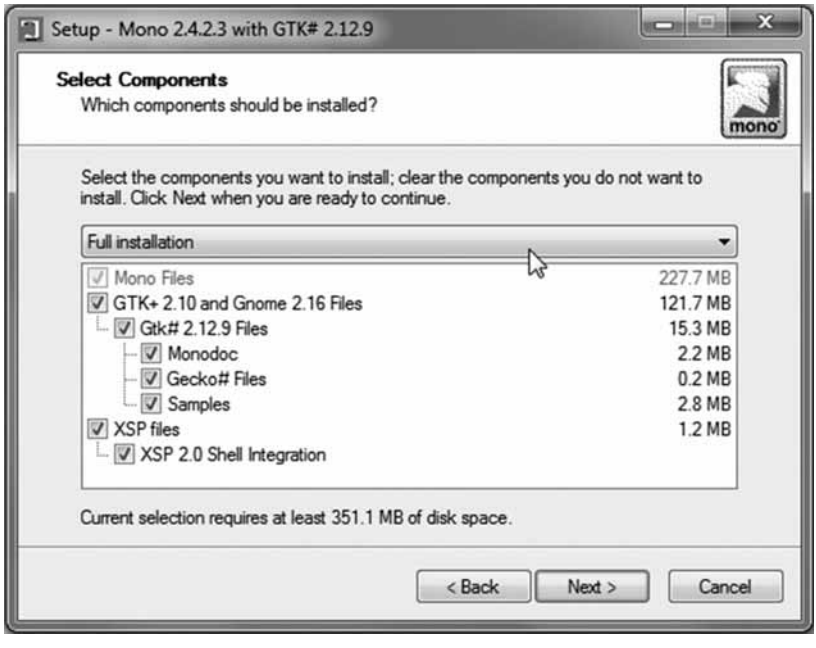

Рис. Б.2. Выбор полной установки Mono

Рис. Б.3. Структура каталогов Mono

Нас будут интересовать только следующие подкаталоги:

- `bin` — содержит большинство инструментов разработки Mono, включая компиляторы C# командной строки;
- `lib\mono\gac` — указывает на местоположение глобального кеша сборок Mono.

Поскольку подавляющее большинство инструментов разработки Mono запускается из командной строки, вам понадобится окно командной строки Mono, которое автоматически распознает все инструменты такого рода. Открыть окно командной строки можно выбором пункта меню Start⇨All Programs⇨Mono <*версия*> for Windows (Пуск⇨Программы⇨Mono <*версия*> для Windows). Для тестирования установки введите следующую команду и нажмите клавишу <Enter>:

```
mono --version
```

Если все в порядке, то вы должны увидеть разнообразную информацию о среде времени выполнения Mono. Вот как может выглядеть вывод:

```
Mono JIT compiler version 2.4.2.3 (tarball)
Copyright (C) 2002-2008 Novell, Inc and Contributors. www.mono-project.com
              TLS:           normal
              GC:            Included Boehm (with typed GC)
              SIGSEGV:       normal
              Notification:  Thread + polling
              Architecture:  x86
              Disabled:      none
```

Языки разработки Mono

Подобно дистрибутиву CLR от Microsoft дистрибутив Mono поставляется с несколькими компиляторами:

- `mcs` — компилятор C# для Mono;
- `vbnc` — компилятор Visual Basic .NET для Mono;
- `booc` — компилятор языка Boo для Mono;
- `ilasm` — компиляторы CIL для Mono.

Хотя внимание здесь сосредоточено только на компиляторах C#, вы должны иметь в виду, что проект Mono содержит и компилятор для Visual Basic (`http://www.mono-project.com/docs/about-mono/languages/visualbasic/`).

Boo — это объектно-ориентированный, статически типизированный язык программирования для CLI, который имеет синтаксис, основанный на Python. Дополнительные сведения о языке программирования Boo можно найти по адресу `http://boo-lang.org/`. Наконец, `ilasm` — это компилятор CIL в Mono.

Работа с компилятором C#

Компилятор C# для проекта Mono называется `mcs`, и он полностью совместим с C# 2008. Как и компилятор C# от Microsoft (`csc.exe`), `mcs` поддерживает файлы ответов, флаг `/target:` (для указания типа сборки), флаг `/out:` (для указания имени скомпилированной сборки) и флаг `/reference:` (для обновления манифеста текущей сборки внешними зависимостями). Список всех опций `mcs` можно получить с помощью команды `mcs -?`.

Создание приложений Mono с использованием MonoDevelop

В результате установки Mono никакой графической IDE-среды не предоставляется. Тем не менее, это вовсе не означает, что вам придется создавать все приложения Mono в командной строке. В дополнение к базовой среде можно также загрузить бесплатную IDE-среду MonoDevelop.

IDE-среда MonoDevelop доступна для загрузки на веб-сайте Mono, и она поддерживает установку пакетов для Mac OS X, разнообразных дистрибутивов Linux, а также Microsoft Windows. После установки вы обнаружите в своем распоряжении интегрированный отладчик, возможности IntelliSense и множество шаблонов проектов (кажем, ASP.NET и Moonlight). Некоторое представление об IDE-среде MonoDevelop позволяет получить рис. Б.4.

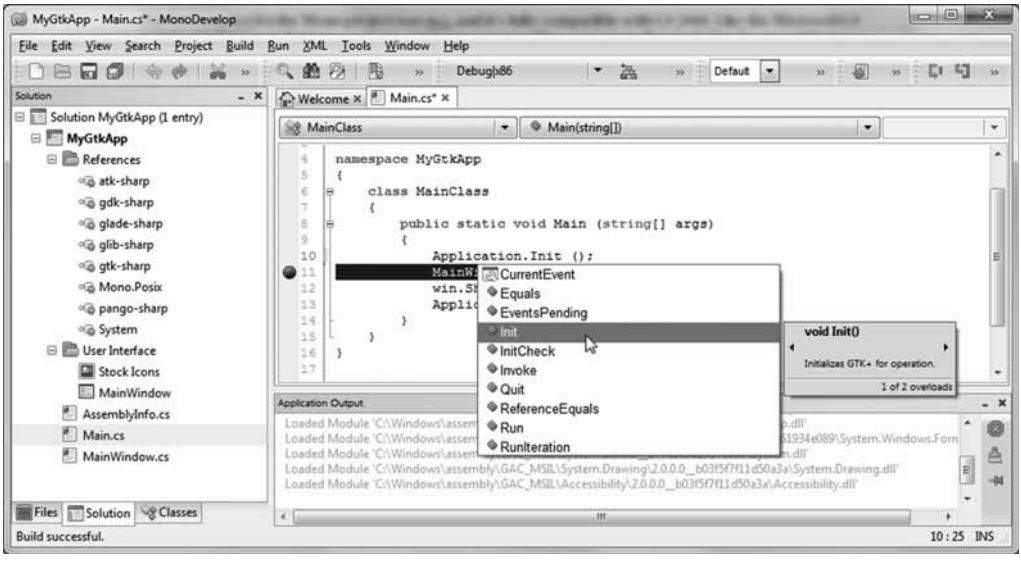

Рис. Б.4. IDE-среда MonoDevelop

Инструменты разработки Mono, совместимые с Microsoft

Вдобавок к управляемым компиляторам Mono поставляется с различными инструментами разработки, которые функционально эквивалентны инструментам, доступным в Microsoft .NET SDK (часть из них имеют те же самые имена).

В табл. Б.3 приведены соответствия между часто применяемыми утилитами Mono и Microsoft .NET.

Инструменты разработки, специфичные для Mono

В состав Mono также входят инструменты разработки, для которых нет прямых эквивалентов в Microsoft .NET Framework SDK (табл. Б.4).

Таблица Б.3. Инструменты командной строки Mono и их аналоги в Microsoft .NET

Утилита Mono	Утилита Microsoft .NET	Описание
`al`	`al.exe`	Манипулирует манифестами сборок и строит многофайловые сборки (помимо прочего)
`gacutil`	`gacutil.exe`	Взаимодействует с GAC
`mono`, когда запускается с опцией `-aot` в качестве начального параметра для выполняющейся сборки	`ngen.exe`	Выполняет предварительную компиляцию кода CIL сборки
`wsdl`	`wsdl.exe`	Генерирует код прокси клиентской стороны для веб-служб XML
`disco`	`disco.exe`	Обнаруживает URL веб-служб XML, находящихся на веб-сервере
`xsd`	`xsd.exe`	Генерирует определения типов из файла схемы XSD
`sn`	`sn.exe`	Генерирует данные ключей для строго именованной сборки
`monodis`	`ildasm.exe`	Дизассемблер CIL
`ilasm`	`ilasm.exe`	Ассемблер CIL
`xsp2`	`webdev.webserver.exe`	Веб-сервер для тестирования и разработки приложений ASP.NET

Таблица Б.4. Инструменты Mono, не имеющие прямых эквивалентов в Microsoft .NET SDK

Инструмент	Описание
`monop`	Утилита `monop` отображает определение указанного типа с использованием синтаксиса C# (краткий пример приведен в следующем разделе)
SQL#	Проект Mono поставляется с графическим интерфейсом (SQL#), который позволяет взаимодействовать с реляционными базами данных посредством разнообразных поставщиков данных ADO.NET
Glade 3	IDE-среда визуальной разработки для построения графических приложений GTK#

На заметку! Инструменты SQL# и Glade можно загрузить, через папку Applications установки Mono. Опробуйте их, чтобы оценить все богатство платформы Mono.

Использование monop

Утилита `monop` (mono print — печать Mono) позволяет отобразить определение C# для заданного типа внутри указанной сборки. Как и можно было ожидать, этот инструмент может быть полезен, когда необходимо быстро просмотреть сигнатуру какого-то метода, не углубляясь в формальную документацию. В качестве примера попробуйте ввести в окне командной строки Mono следующую команду:

```
monop System.Object
```

Вы должны увидеть определение для хорошо известного типа `System.Object`:

```
[Serializable]
public class Object {
  public Object ();
  public static bool Equals (object objA, object objB);
  public static bool ReferenceEquals (object objA, object objB);
  public virtual bool Equals (object obj);
  protected override void Finalize ();
  public virtual int GetHashCode ();
  public Type GetType ();
  protected object MemberwiseClone ();
  public virtual string ToString ();
}
```

Построение приложений .NET с помощью Mono

Давайте посмотрим на Mono в действии. Создайте библиотеку кода по имени `CoreLibDumper.dll`. Эта сборка содержит единственный тип класса `CoreLibDumper`, который поддерживает статический метод `DumpTypeToFile()`. Метод принимает строковый параметр, который представляет полностью заданное имя любого типа внутри `mscorlib.dll`, и получает информацию о типе с помощью API-интерфейса рефлексии, выводя сигнатуры членов классов в локальный файл на жестком диске.

Построение библиотеки кода Mono

Создайте на диске C: новую папку с именем `MonoCode`. Внутри нее создайте подпапку по имени `CorLibDumper`, а в ней файл кода C# с именем `CorLibDumper.cs` и следующим содержимым:

```
// CorLibDumper.cs
using System;
using System.Reflection;
using System.IO;

// Определить версию сборки.
[assembly:AssemblyVersion("1.0.0.0")]

namespace CorLibDumper
{
  public class TypeDumper
  {
    public static bool DumpTypeToFile(string typeToDisplay)
    {
      // Попробовать загрузить тип в память.
      Type theType = null;
      try
      {
        // Второй параметр GetType() указывает, нужно ли
        // генерировать исключение, если тип не найден.
        theType = Type.GetType(typeToDisplay, true);
      } catch { return false; }
```

```
      // Создать локальный файл *.txt.
      using(StreamWriter sw =
         File.CreateText(string.Format("{0}.txt", theType.FullName)))
      {
        // Записать информацию о типе в файл.
        sw.WriteLine("Type Name: {0}", theType.FullName);
        sw.WriteLine("Members:");
        foreach(MemberInfo mi in theType.GetMembers())
          sw.WriteLine("\t-> {0}", mi.ToString());
      }
      return true;
    }
  }
}
```

Подобно компилятору C# от Microsoft компиляторы Mono поддерживают применение файлов ответов. Хотя файл можно было бы скомпилировать, указывая вручную все требуемые аргументы в командной строке, лучше создать новый файл по имени `LibraryBuild.rsp` (там же, где находится `CorLibDumper.cs`) со следующим набором команд:

```
/target:library
/out:CorLibDumper.dll
CorLibDumper.cs
```

Теперь библиотеку можно скомпилировать в командной строке:

```
mcs @LibraryBuild.rsp
```

Подход функционально эквивалентен следующей (более многословной) команде:

```
mcs /target:library /out:CorLibDumper.dll CorLibDumper.cs
```

Назначение строгого имени библиотеке `CoreLibDumper.dll`

Платформа Mono поддерживает понятие развертывания строго именованных сборок в Mono GAC. Для генерации необходимых данных открытого и секретного ключей Mono предоставляет утилиту командной строки `sn`, которая работает более или менее идентично инструменту Microsoft с тем же самым именем. Например, приведенная ниже команда генерирует новый файл `*.snk` (все возможные аргументы можно просмотреть посредством опции `-?`):

```
sn -k myTestKeyPair.snk
```

Сообщить компилятору C# о необходимости использования этих данных ключей для назначения строгого имени библиотеке `CorLibDumper.dll` можно, добавив в файл `LibraryBuild.rsp` дополнительную команду:

```
/target:library
/out:CorLibDumper.dll
/keyfile:myTestKeyPair.snk
CorLibDumper.cs
```

Теперь перекомпилируйте сборку:

```
mcs @LibraryBuild.rsp
```

Просмотр измененного манифеста с помощью monodis

Перед развертыванием сборки в Mono GAC вы должны освоить утилиту командной строки monodis, которая функционально эквивалентна утилите ildasm.exe от Microsoft (но не имеет графического интерфейса). С применением monodis можно просматривать код CIL, манифест и метаданные типов для указанной сборки. Нас интересуют основные детали, связанные с (уже строго именованной) сборкой, для чего используется флаг --assembly. Результат выполнения следующей команды показан на рис. Б.5:

```
monodis --assembly CorLibDumper.dll
```

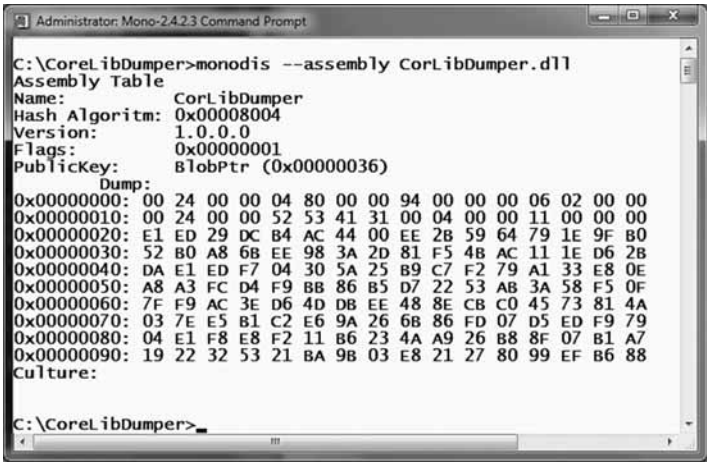

Рис. Б.5. Просмотр кода CIL, метаданных и манифеста сборки с помощью утилиты monodis

Манифест сборки теперь содержит открытый ключ, определенный в файле myTestKeyPair.snk.

Установка сборок в Mono GAC

К этому моменту сборка CorLibDumper.dll снабжена строгим именем и теперь ее можно установить в Mono GAC. Подобно одноименному инструменту от Microsoft утилита gacutil в Mono поддерживает установку, удаление и вывод списка текущих сборок, установленных в C:\Program Files\Mono-<*версия*>\lib\mono\gac. Приведенная далее команда развертывает сборку CorLibDumper.dll в GAC и регистрирует ее как разделяемую сборку на машине:

```
gacutil -i CorLibDumper.dll
```

На заметку! Удостоверьтесь, что для установки этого двоичного файла в Mono GAC применяется окно командной строки Mono. В случае использования утилиты gacutil.exe от Microsoft сборка CorLibDumper.dll будет установлена в Microsoft GAC!

Открыв каталог \gac после выполнения этой команды, вы должны увидеть новую папку CorLibDumper (рис. Б.6). Внутри нее находится подкаталог, который следует тем же соглашениям по именованию, что и Microsoft GAC (*версияСборки_маркерОткрытогоКлюча*).

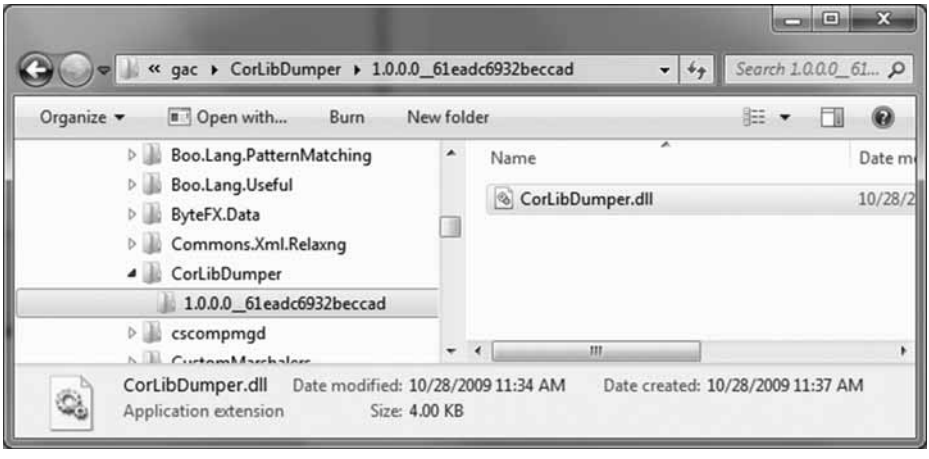

Рис. Б.6. Развертывание библиотеки кода в Mono GAC

На заметку! Опция `-l` утилиты `gacutil` позволяет вывести список всех сборок в Mono GAC.

Построение консольного приложения в Mono

Вашим первым клиентом Mono будет простое консольное приложение по имени `ConsoleClientApp.exe`. Создайте в папке `C:\MonoCode\CorLibDumper` новый файл `ConsoleClientApp.cs` со следующим определением класса `Program`:

```
// Это клиентское приложение использует CorLibDumper.dll
// для вывода информации о типе в файл.
using System;
using CorLibDumper;
namespace ConsoleClientApp
{
  public class Program
  {
    public static void Main(string[] args)
    {
      Console.WriteLine("***** The Type Dumper App *****\n");
      // Запросить имя типа у пользователя.
      string typeName = "";
      Console.Write("Please enter type name: ");
      typeName = Console.ReadLine();
      // Передать его вспомогательной библиотеке.
      if(TypeDumper.DumpTypeToFile(typeName))
        Console.WriteLine("Data saved into {0}.txt", typeName);
      else
        Console.WriteLine("Error! Can't find that type...");
                        // Ошибка! Тип не найден.
    }
  }
}
```

Обратите внимание, что метод `Main()` запрашивает у пользователя полностью заданное имя типа. Метод `TypeDumper.DumpTypeToFile()` применяет введенное пользователем имя для вывода членов типа в локальный файл. Создайте файл `ClientBuild.rsp` для этого клиентского приложения и укажите ссылку на `CorLibDumper.dll`:

```
/target:exe
/out:ConsoleClientApp.exe
/reference:CorLibDumper.dll
ConsoleClientApp.cs
```

В окне командной строки Mono скомпилируйте приложение с помощью `mcs`:

```
mcs @ClientBuild.rsp
```

Загрузка клиентского приложения в исполняющую среду Mono

Теперь приложение `ConsoleClientApp.exe` можно загрузить в исполняющую среду Mono, указав в качестве аргумента имя исполняемого файла (с расширением `*.exe`):

```
mono ConsoleClientApp.exe
```

В ответ на запрос введите `System.Threading.Thread` и нажмите клавишу <Enter>. Вы увидите новый файл по имени `System.Threading.Thread.txt`, который содержит определение метаданных типа (рис. Б.7).

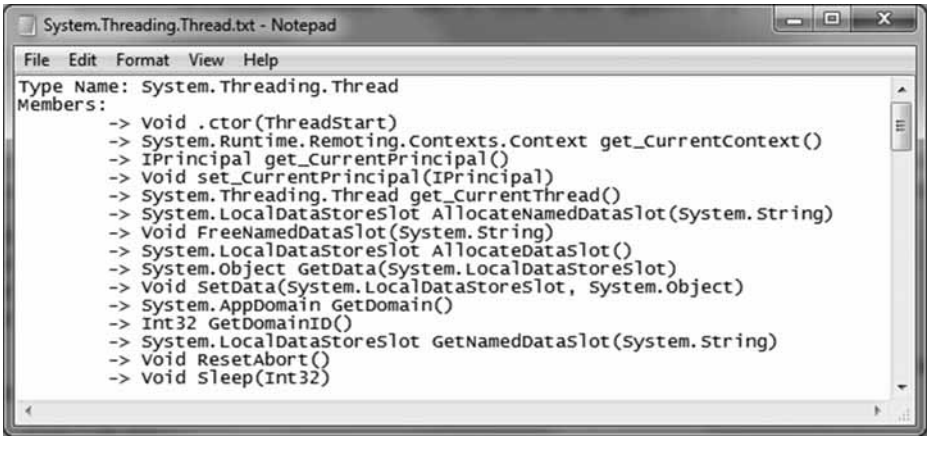

Рис. Б.7. Результаты выполнения клиентского приложения

Прежде чем перейти к созданию клиента Windows Forms, проведите следующий эксперимент. Измените имя сборки `CorLibDumper.dll` в папке, содержащей клиентское приложение, на `DontUseCorLibDumper.dll`. Клиентское приложение попрежнему должно успешно запускаться, т.к. доступ к данной сборке при создании клиента необходим только для изменения манифеста клиента. Во время выполнения исполняющая среда Mono загрузит версию `CorLibDumper.dll`, развернутую в Mono GAC.

Однако если вы попытаетесь запустить приложение `ConsoleClientApp.exe` двойным щелчком в проводнике Windows, то столкнетесь с исключением

FileNotFoundException. На первый взгляд может показаться, что оно возникло из-за переименования сборки CorLibDumper.dll в папке клиентского приложения. Тем не менее, истинной причиной является загрузка ConsoleClientApp.exe в среду Microsoft CLR!

Чтобы запустить приложение под управлением Mono, потребуется передать его исполняющей среде Mono посредством команды mono. В противном случае сборка будет загружена в Microsoft CLR, что предполагает установку всех разделяемых сборок в Microsoft GAC внутри каталога <%windir%>\Assembly.

На заметку! Если вы дважды щелкнете на исполняемом файле в проводнике Windows, то сборка будет загружена в Microsoft CLR, а не в исполняющую среду Mono!

Построение клиентской программы Windows Forms

Переименуйте файл DontUseCorLibDumper.dll снова в CorLibDumper.dll, чтобы можно было скомпилировать следующий пример. Создайте новый файл кода C# по имени WinFormsClientApp.cs и сохраните его вместе с файлами текущего проекта. Определите в новом файле два класса, как показано ниже:

```
using System;
using System.Windows.Forms;
using CorLibDumper;
using System.Drawing;

namespace WinFormsClientApp
{
  // Объект Application.
  public static class Program
  {
    public static void Main(string[] args)
    {
      Application.Run(new MainWindow());
    }
  }
  // Простое окно.
  public class MainWindow : Form
  {
    private Button btnDumpToFile = new Button();
    private TextBox txtTypeName = new TextBox();
    public MainWindow()
    {
      // Настроить пользовательский интерфейс.
      ConfigControls();
    }
    private void ConfigControls()
    {
      // Сконфигурировать форму.
      Text = "My Mono Win Forms App!";
      ClientSize = new System.Drawing.Size(366, 90);
      StartPosition = FormStartPosition.CenterScreen;
      AcceptButton = btnDumpToFile;
```

```
      // Сконфигурировать кнопку.
      btnDumpToFile.Text = "Dump";
      btnDumpToFile.Location = new System.Drawing.Point(13, 40);
      // Анонимно обработать событие щелчка.
      btnDumpToFile.Click += delegate
      {
        if(TypeDumper.DumpTypeToFile(txtTypeName.Text))
          MessageBox.Show(string.Format(
            "Data saved into {0}.txt",
            txtTypeName.Text));
        else
          MessageBox.Show("Error!  Can't find that type...");
      };
      Controls.Add(btnDumpToFile);
      // Сконфигурировать текстовое поле.
      txtTypeName.Location = new System.Drawing.Point(13, 13);
      txtTypeName.Size = new System.Drawing.Size(341, 20);
      Controls.Add(txtTypeName);
    }
  }
}
```

Чтобы скомпилировать это приложение Windows Forms, используя файл ответов, создайте файл по имени `WinFormsClientApp.rsp` и поместите в него следующие команды:

```
/target:winexe
/out:WinFormsClientApp.exe
/r:CorLibDumper.dll
/r:System.Windows.Forms.dll
/r:System.Drawing.dll
WinFormsClientApp.cs
```

Удостоверившись, что этот файл сохранен в той же самой папке, что и код примера, передайте его в качестве аргумента компилятору `mcs` (с помощью `@`):

```
msc @WinFormsClientApp.rsp
```

Наконец, запустите полученное приложение Windows Forms с применением Mono:

```
mono WinFormsClientApp.exe
```

Результаты тестового запуска показаны на рис. Б.8.

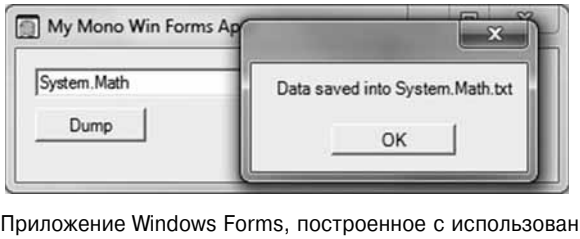

Рис. Б.8. Приложение Windows Forms, построенное с использованием Mono

Выполнение приложения Windows Forms под управлением Linux

До сих пор мы демонстрировали создание сборок, которые можно было бы скомпилировать и с помощью Microsoft .NET Framework SDK. Однако важность Mono станет яснее, если вы взглянете на рис. Б.9, где показано то же самое приложение Windows Forms, функционирующее под управлением SuSe Linux. Обратите внимание, что приложение Windows Forms имеет корректный внешний вид (с учетом текущей темы рабочего стола).

Рис. Б.9. Выполнение приложения Windows Forms под SuSe Linux

Исходный код. Проект `CorLibDumper` находится в подкаталоге `Appendix_B`.

Предшествующие примеры показали, что точно такой же код C# можно компилировать и выполнять под управлением Linux (а также любой другой операционной системы, поддерживаемой Mono) с применением тех же самых инструментов разработки Mono. На самом деле любую сборку, созданную без использования программных конструкций .NET 4.0, можно развернуть или перекомпилировать для другой операционной системы, совместимой с Mono, и затем запустить с помощью утилиты времени выполнения `mono`. Поскольку все сборки содержат код CIL, не зависящий от платформы, приложения вообще не придется компилировать заново.

Кто использует Mono?

В этом кратком приложении на нескольких простых примерах были продемонстрированы возможности платформы Mono. Если вы собираетесь создавать программы .NET только для семейства операционных систем Windows, то вам могут и не встретиться компании или отдельные разработчики, которые активно применяют Mono. Невзирая на это, Mono интенсивно используется в сообществах программистов для систем Mac OS X, Linux и Windows.

Например, зайдите в раздел Software (Программное обеспечение) веб-сайта Mono:

`http://www.mono-project.com/docs/about-mono/showcase/software/`

Вы обнаружите длинный список коммерческих продуктов, построенных с использованием Mono, включая инструменты разработки, серверные продукты, видеоигры (в том числе для Nintendo Wii и iPhone) и медицинские прикладные системы.

Пройдя по ссылкам, вы увидите, что платформа Mono полностью оснащена для создания по-настоящему межплатформенного программного обеспечения .NET производственного уровня.

Рекомендации по дальнейшему изучению

Если вы внимательно проработали весь материал, изложенный в книге, то уже много чего знаете о Mono, т.к. это реализация CLI, совместимая с ECMA. Чтобы узнать больше об особенностях Mono, лучше всего начинать с официального веб-сайта Mono (`www.mono-project.com`). В частности, страница `www.mono-project.com/archived/use/` может послужить отправной точкой для нескольких важных тем, включая доступ к базам данных с помощью ADO.NET, разработку веб-приложений с применением ASP.NET и т.д.

Кроме того, на веб-сайте DevX (`www.devx.com`) есть пара статей, которые могут вас заинтересовать.

- *Mono IDEs: Going Beyond the Command Line* (IDE-среды Mono: выходим за рамки командной строки) — в этой статье рассматриваются многие IDE-среды, пригодные для разработки приложений Mono.
- *Building Robust UIs in Mono with Gtk#* (Построение надежных пользовательских интерфейсов в Mono с помощью Gtk#) — здесь описано создание настольных приложений с использованием инструментального набора GTK# как альтернативы Windows Forms.

Вдобавок вы должны освоиться с веб-сайтом документации Mono (`docs.go-mono.com`). На нем находится документация по библиотекам базовых классов Mono, инструментам разработки и другим темам (рис. Б.10).

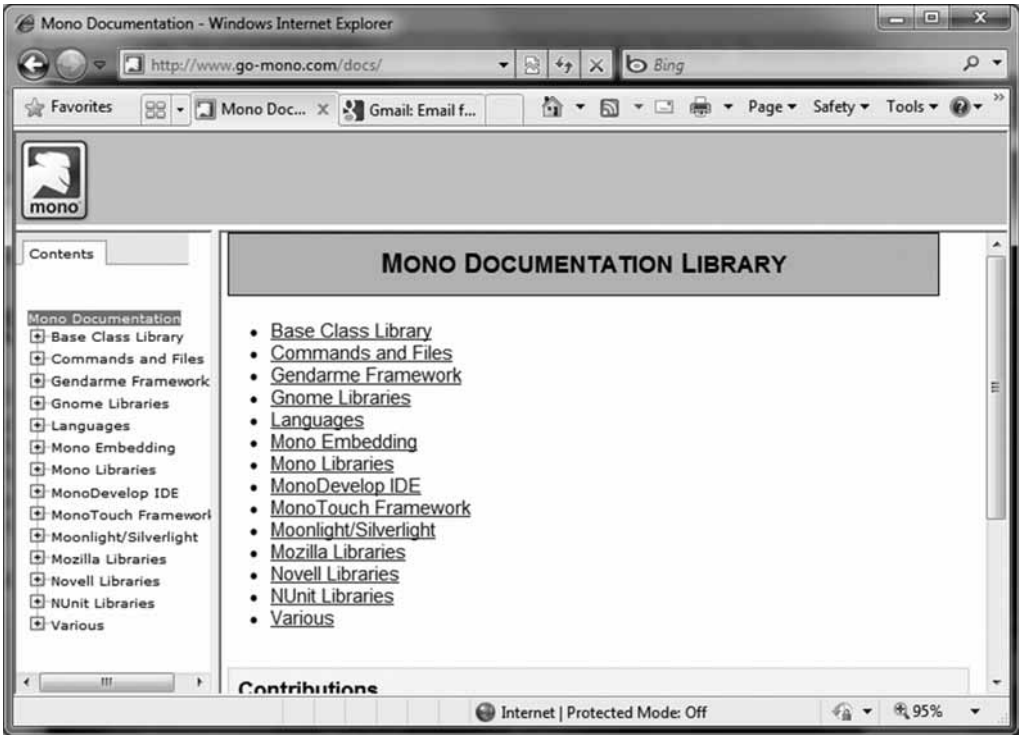

Рис. Б.10. Онлайновая документация Mono

На заметку! Веб-сайт с онлайновой документацией Mono поддерживается сообществом разработчиков, поэтому не удивляйтесь, если обнаружите ссылки на незавершенные документы! Учитывая, что Mono является совместимым с ECMA дистрибутивом Microsoft .NET, во время исследований Mono можно работать с полнофункциональной онлайновой документацией MSDN.

Резюме

Целью настоящего приложения было введение в межплатформенную природу языка программирования C# и платформы .NET во время применения инфраструктуры Mono. Вы узнали, что Mono поставляется с несколькими инструментами командной строки, которые позволяют строить разнообразные сборки .NET, включая строго именованные сборки, разворачиваемые в GAC, приложения Windows Forms и библиотеки кода .NET. Если вам необходимо создавать приложения .NET, которые должны выполняться под управлением разных операционных систем, то проект Mono будет великолепным вариантом.