ПРИЛОЖЕНИЕ Б

Независимая от платформы разработка .NET-приложений с помощью Mono

В настоящем приложении вы ознакомитесь с межплатформенной разработкой приложений на C# и .NET, используя реализацию .NET с открытым исходным кодом, которая называется *Mono*. Если вам интересно, что это значит — по-испански *mono* означает "обезьяна", и это указание на различные символы-талисманы платформы Mono, которые использовались первоначальными разработчиками — корпорацией Ximian. Вы узнаете о роли общеязыковой инфраструктуры, общей области применения Mono и различных инструментах разработки Mono. Когда вы прочтете это приложение, вы сможете при желании самостоятельно углублять свои навыки в разработке Mono-приложений.

На заметку! В качестве подробного описания межплатформенной разработки в .NET рекомендуется обратиться к книге Марка Истона и Джейсона Кинга *Cross-Platform .NET Development: Using Mono, Portable .NET, and Microsoft .NET* (Apress, 2004 г.).

Платформенная независимость .NET

Когда-то программистам, которые использовали язык разработки Microsoft (например, VB6) или среду программирования Microsoft (такую как MFC, COM или ATL), приходилось ограничиваться только созданием программного обеспечения, которое (обычно) работало лишь под управлением семейства операционных систем Windows. Многие разработчики .NET, привыкшие к такой привязке к Microsoft, бывают удивлены, когда узнают, что система .NET *не зависит от платформы*. Тем не менее, это так. Создавать, компилировать и выполнять сборки .NET можно под управлением операционных систем, отличных от Microsoft.

С помощью реализаций .NET с открытым исходным кодом, таких как Mono, приложения .NET могут функционировать в средах многих других операционных систем, включая Mac OS X, Solaris, AIX и многочисленные разновидности Unix/Linux. Mono также предоставляет установочный пакет для Microsoft Windows. Таким образом, можно создавать и выполнять сборки .NET под управлением Windows, даже не устанавливая Microsoft .NET Framework 4.0 SDK или Visual Studio.

На заметку! Следует отметить, что лучшими средствами создания программ .NET для семейства операционных систем Windows являются Microsoft .NET Framework 4.0 SDK и Visual Studio.

Даже когда разработчики узнают о межплатформенных возможностях кода .NET, они часто считают, что область независимой от платформы разработки .NET-приложений ограничена консольными приложениями на уровне "Hello world". Однако на самом деле можно строить сборки производственного уровня, в которых используются ADO.NET, Windows Forms (в дополнение к альтернативным пакетам создания графических пользовательских интерфейсов наподобие GTK# и Cocoa#), ASP.NET, LINQ и веб-службы XML, работая со многими основными пространствами имен и языковыми возможностями.

Роль CLI

Реализация межплатформенных возможностей .NET отличается от подхода, принятого Sun Microsystems в отношении платформы программирования на языке Java. В отличие от Java, компания Microsoft не разрабатывает программы установки .NET для Mac, Linux и т.д. Вместо этого Microsoft выпустила набор формализованных спецификаций, которые могут использоваться другими субъектами в качестве дорожной карты для создания дистрибутивов .NET для их платформ. Все вместе эти спецификации называются *общеязыковой инфраструктурой* (Common Language Infrastructure — CLI).

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

Какая польза от этого нам? Эти две спецификации предоставляют дорожную карту для компаний, разработчиков, университетов и других организаций, которые хотят создавать самостоятельно разработанные дистрибутивы языка программирования C# и платформы .NET. Упомянутые спецификации — это:

- *ECMA-334* — определяет синтаксис и семантику языка программирования C#;
- *ECMA*-335 — определяет многие детали платформы .NET, которые все вместе называются *общеязыковой инфраструктурой*.

Спецификация ECMA-334 определяет лексику и грамматику C# очень формализованным научным способом (понятно, что такой уровень детализации весьма важен для тех, кто хочет реализовать компилятор для C#). ECMA-335 имеет гораздо больший объем и поэтому поделена на шесть разделов (см. табл. Б.1).

Таблица Б.1. Разделы спецификации ECMA-335

Раздел ECMA-335	Описание
I. Концепции и архитектура	Описание общей архитектуры CLI: правила системы общих типов, спецификация общего языка и принципы работы механизма времени выполнения .NET
II. Определение метаданных и семантики	Подробное описание формата метаданных .NET
III. Набор инструкций CIL	Описание синтаксиса и семантики общего промежуточного языка программирования (CIL)
IV. Профили и библиотеки	Высокоуровневый обзор минимального и полного набора библиотек классов, которые должны поддерживаться дистрибутивом .NET, совместимым с CLI

Окончание табл. Б.1

Раздел ECMA-335	Описание
V. Форматы обмена отладочной информацией	Описание переносимого формата обмена отладочной информацией (CILDB). Переносимые файлы CILDB обеспечивают стандартный способ обмена отладочной информацией между производителями и потребителями CLI
VI. Приложения	Разнообразные статьи, которые проясняют такие темы, как рекомендации по созданию библиотек классов и детали реализации компилятора CIL

В данном приложении мы не будем вдаваться в подробности спецификаций ECMA-334 и ECMA-335 — впрочем, чтобы уметь создавать независимые от платформы сборки .NET, вы и не должны разбираться во всех тонкостях этих документов. Но если вам интересна эта тема, можете загрузить обе спецификации с веб-сайта ECMA (`http://www.ecma-international.org/publications/standards/Standard.htm`).

Главные дистрибутивы CLI

В настоящее время помимо CLR, разработанной Microsoft, существуют две главные реализации CLI– это 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`	Дистрибутив .NET с открытым исходным кодом и коммерческой поддержкой, спонсируемый корпорацией Novell. Предназначен для работы под множеством популярных разновидностей Unix/Linux, Mac OS X, Solaris и Windows
Portable .NET	`www.dotgnu.org`	Распространяется по универсальной общественной лицензии GNU. Как понятно из названия, Portable .NET предназначен для работы на множестве разнообразных операционных систем и архитектур, в том числе и таких экзотических, как BeOS, Microsoft Xbox и Sony PlayStation

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

Кроме реализаций основных библиотек, определенных в разделе IV спецификации ECMA-335, в Mono и Portable .NET имеются 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).

В состав дистрибутивов Mono и Portable .NET входит также ряд сборок, предназначенных конкретно для работы с операционными системами Unix/Linux и Mac OS X. Например, Cocoa# является оболочкой .NET для инструментального набора Cocoa, широко применяющегося для разработки графических пользовательских интерфейсов Mac OX. В этом приложении будут обсуждаться только технологии программирования, не зависящие от конкретной операционной системы.

На заметку! В этом приложении не рассматривается дистрибутив Portable .NET; тем не менее, важно знать, что Mono не является единственным независимый от платформы дистрибутивом .NET, доступным в настоящее время. Рекомендуется, кроме знакомства с платформой Mono, самостоятельно поэкспериментировать и с Portable .NET.

Область действия Mono

Понятно, что если Mono является API-интерфейсом, построенным на существующих спецификациях ECMA, которые были созданы в Microsoft, то Mono постоянно обновляется с выходом новых версий платформы Microsoft .NET. Эта технология позволяет создавать веб-сайты ASP.NET, приложения Windows Forms, приложения работы с базами данных с помощью ADO.NET и (конечно) простые консольные приложения.

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

Mono также поддерживает некоторые технологии на основе .NET, у которых нет прямых аналогов от Microsoft. К примеру, Mono поставляется с GTK# — оболочкой .NET для популярной среды разработки графических пользовательских интерфейсов в Linux под названием GTK. Привлекательным API-интерфейсом на основе Mono является MonoTouch, позволяющий создавать приложения для устройств Apple iPhone и iTouch на языке программирования C#.

На заметку! На веб-сайте Mono имеется страница с описанием общей функциональности Mono и планами на разработку последующих выпусков (`www.mono-project.com/plans`).

И еще один интересный момент, касающийся возможностей Mono: подобно .NET Framework 4.0 SDK от Microsoft, Mono SDK также поддерживает несколько языков программирования для .NET. В данном приложении используется C#, однако в Mono имеется поддержка компилятора Visual Basic, а также многих других языков программирования, ориентированных на работу с .NET.

Получение и установка Mono

Теперь, когда вы имеете лучшее представление о возможностях платформы Mono, рассмотрим вопрос получения и установки самой Mono. Зайдите на веб-сайт Mono (`www.mono-project.com`) и перейдите на вкладку Download (Загрузка). Здесь доступны для загрузки различные программы установки (рис. Б.1).

Здесь будет описан процесс установки Windows-дистрибутива Mono (эта установка никак не повлияет на любую существующую установку Microsoft .NET или Visual Studio). Загрузите последний стабильный установочный пакет для Microsoft Windows и сохраните установочную программу на локальный жесткий диск.

При выполнении установочной программы вы сможете установить различные средства разработки Mono, кроме стандартных библиотек базовых классов и средств программирования на C#. А именно, программа установки спросит, хотите ли вы установить GTK# (API-интерфейс .NET для разработки графических пользовательских интерфейсов с открытым исходным кодом на основе предназначенного для Linux инструментального набора GTK) и XSP (автономный веб-сервер, подобный веб-серверу разработки ASP.NET от Microsoft — `webdev.webserver.exe`).

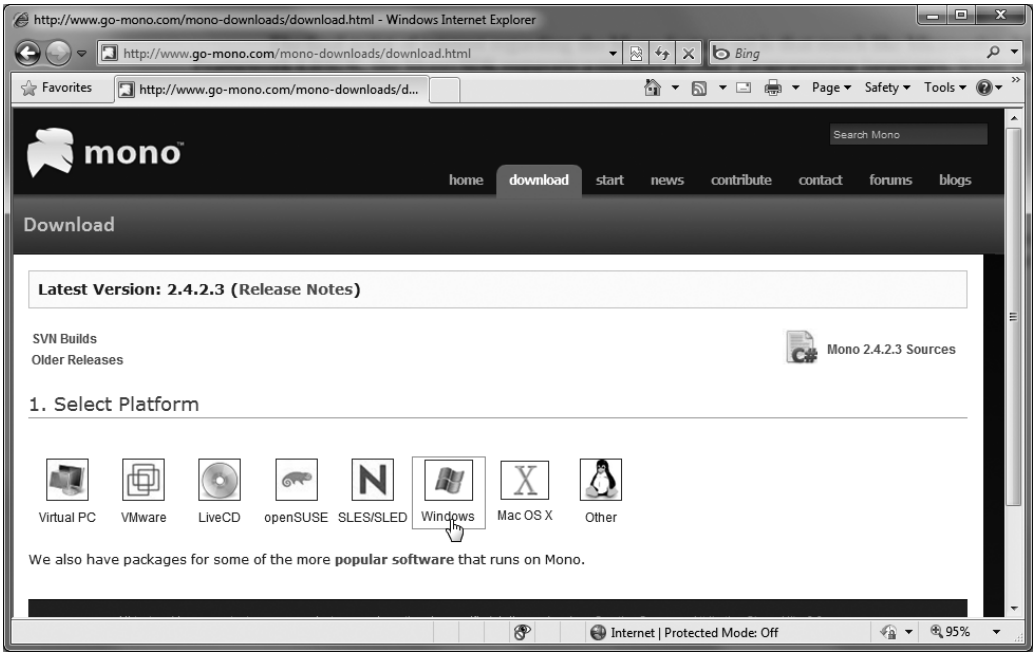

Рис. Б.1. Страница загрузки Mono

Здесь мы будем считать, что вы выбрали вариант, который отмечает все пункты в сценарии установки (рис. Б.2).

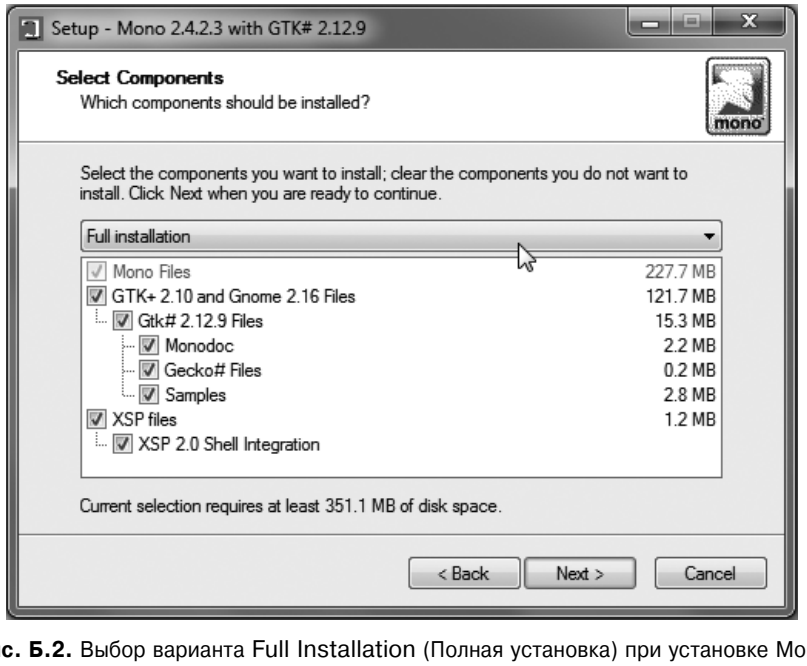

Рис. Б.2. Выбор варианта Full Installation (Полная установка) при установке Mono

Структура каталога Mono

По умолчанию Mono устанавливается в каталог `C:\Program Files\Mono-<версия>` (на момент написания этих строк последней версией Mono была 2.4.2.3, но ваш номер версии почти наверняка будет другим). В этом каталоге находятся несколько подкаталогов (рис. Б.3).

Рис. Б.3. Структура каталога Mono

В данном приложении нас будут интересовать только следующие подкаталоги:

- `bin` — содержит большинство средств разработки Mono, в том числе и компиляторы C# уровня командной строки;
- `lib\mono\gac` — указывает на местоположение глобального кеша сборок Mono.

Поскольку большинство средств разработки Mono запускаются из командной строки, вам понадобится командная строка Mono, которая автоматически распознает все такие средства. Открыть командную строку (функционально эквивалентную командной строке разработчика в Visual Studio) можно с помощью меню Start All Programs Mono <версия> For Windows (Пуск⇨Программы⇨Mono *<версия>* для Windows). Для проверки установки введите следующую команду и нажмите клавишу <Enter>:

```
mono --version
```

Если все в порядке, вы должны увидеть разнообразную информацию о среде времени выполнения Mono. Вот данные, полученные на моем компьютере с установленной платформой Mono:

```
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

Как и в дистрибутиве CLR от Microsoft, 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.

В данном приложении рассматриваются только компиляторы для C#, однако помните, что проект Mono содержит и компилятор для Visual Basic (`www.mono-project.com/Visual_Basic`).

Boo — это объектно-ориентированный язык программирования со статическими типами для CLI, поддерживающий синтаксис на основе языка Python. Дополнительные сведения о языке программирования Boo можно найти на сайте `http://boo.codehaus.org`. Наконец, `ilasm` — это компилятор CIL для платформы 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 с внешними зависимостями). Список всех опций `mcs` можно получить с помощью следующей команды:

```
mcs -?
```

Создание приложений Mono с помощью MonoDevelop

После установки Mono графическая IDE-среда не предоставляется. Однако это совсем не означает, что вам придется создавать все приложения Mono с помощью командной строки. Кроме базовой среды, можно дополнительно загрузить бесплатную IDE-среду MonoDevelop. Как понятно из названия, MonoDevelop создана с помощью кодовой базы SharpDevelop.

IDE-среду MonoDevelop можно загрузить с веб-сайта Mono, и она поддерживает установку пакетов для Mac OS X, различных дистрибутивов Linux, а также Microsoft Windows. После установки вы обнаружите в своем распоряжении интегрированный отладчик, возможности IntelliSense и множество шаблонов проектов (таких как ASP.NET и Moonlight). Некоторое понятие о то, что собой представляет IDE-среда MonoDevelop, можно получить на рис. Б.4.

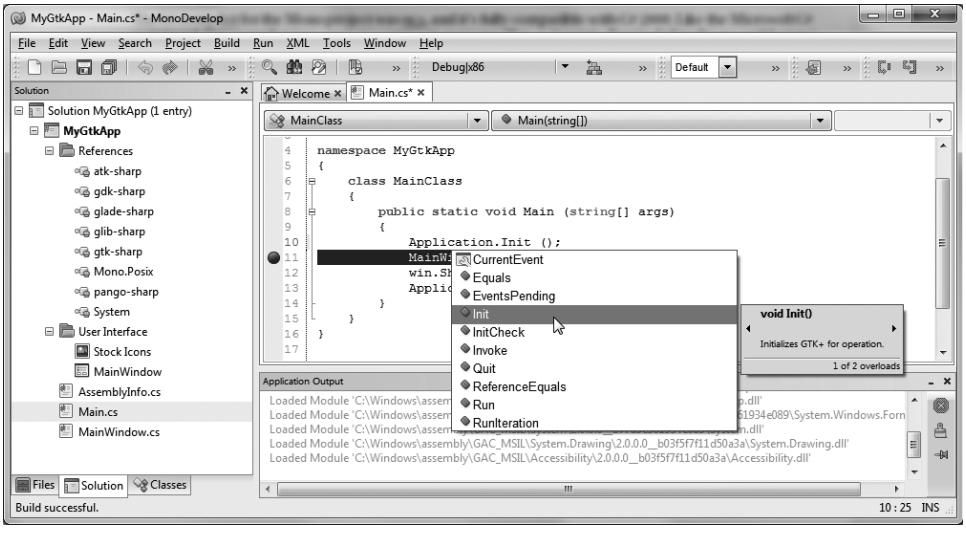

Рис. Б.4. IDE-среда MonoDevelop

Средства разработки в Mono, совместимые с Microsoft

Кроме управляемых компиляторов, в состав Mono входят различные средства разработки, функционально эквивалентные свойствам, которые доступны в Microsoft .NET SDK (некоторые из них имеют те же имена). В табл. Б.3 приведены соответствия между некоторыми часто используемыми утилитами Mono и Microsoft .NET.

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

Утилита Mono	Утилита Microsoft .NET	Описание
`al`	`al.exe`	Обработка манифестов сборок и создание сборок из нескольких файлов (и кое-что еще)
`gacutil`	`gacutil.exe`	Взаимодействие с GAC
`Mono` при запуске с опцией `-aot` при работающей сборке	`ngen.exe`	Предварительная компиляция CIL-кода сборки
`wsdl`	`wsdl.exe`	Генерация кода для модуля доступа на стороне клиента для веб-служб на основе XML
`disco`	`disco.exe`	Обнаружение URL-адресов веб-служб XML, которые находятся на веб-сервере
`xsd`	`xsd.exe`	Генерация определений типов из XSD-схемы файла
`sn`	`sn.exe`	Генерация ключевых данных для строго именованной сборки
`monodis`	`ildasm.exe`	Дизассемблер CIL
`ilasm`	`ilasm.exe`	Ассемблер CIL
`xsp2`	`webdev.webserver.exe`	Веб-сервер для тестирования и разработки ACP.NET-приложений

Свойства разработки, имеющиеся только в Mono

Кроме этого, в состав Mono входят инструменты разработки, для которых нет прямых эквивалентов в Microsoft .NET Framework 3.5 SDK (табл. Б.4).

Таблица Б.4. Инструменты Mono, для которых нет прямых эквивалентов в Microsoft .NET SDK

Средство разработки из Mono	Описание
monop	Утилита `monop` выводит определение заданного типа с помощью синтаксиса C# (краткий пример приведен в следующем разделе)
SQL#	Mono Project поставляется с графическим интерфейсом (`SQL#`), который позволяет взаимодействовать с реляционными базами данных с помощью различных поставщиков данных ADO.NET
Glade 3	IDE-среда визуальной разработки для создания графических приложений GTK#

На заметку! SQL# и Glade можно загрузить, щелкнув на кнопке Пуск в Windows и перейдя в папку Applications установки Mono. Попробуйте — это наглядно демонстрирует богатство платформы Mono.

Использование monop

Утилита `monop` (mono print — печать Mono) применяется для вывода определения C# для данного типа в указанной сборке. Понятно, что этот инструмент может быть весьма полезным, если нужно быстро просмотреть сигнатуру какого-либо метода, вместо того чтобы искать ее в формальной документации. В качестве примера попробуйте ввести в командной строке Mono следующую команду:

```
monop System.Object
```

Вы увидите определение для хорошо известного вам 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 ();
}
```

Создание приложений .NET с помощью Mono

А сейчас опробуем Mono в действии. Вначале создайте библиотеку кода под названием `CoreLibDumper.dll`. Эта сборка содержит единственный тип класса с именем `CoreLibDumper`, который поддерживает статический метод `DumpTypeToFile()`. Данный метод принимает строковый параметр — полностью заданное имя любого типа из `mscorlib.dll` — и получает соответствующую информацию о типе с помощью API-интерфейса рефлексии, а затем выводит сигнатуры классов-членов в локальный файл на жестком диске.

Создание библиотеки кода Mono

Создайте на диске C: новую папку с именем MonoCode. В этой папке 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)
    {
      // Попытаться 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)))
      {
        // Выгрузить тип 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;
    }
  }
}
```

Как и компилятор C# от Microsoft, компиляторы Mono поддерживают использование файлов ответов. Этот файл можно скомпилировать, указав вручную все нужные аргументы в командной строке, но проще создать новый файл по имени LibraryBuild.rsp (в том же каталоге, что и CorLibDumper.cs) со следующим набором команд:

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

Теперь для компиляции библиотеки из командной строки понадобится ввести такую команду:

```
mcs @LibraryBuild.rsp
```

Этот подход функционально эквивалентен следующей (более объемной) команде:

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

Назначение библиотеке CoreLibDumper.dll строгого имени

В Mono поддерживается развертывание строго именованных сборок в Mono GAC. Для генерации необходимых открытых и закрытых ключевых данных в Mono имеется утилита командной строки sn, которая работает примерно так же, как средство от Microsoft с тем же именем. Например, следующая команда генерирует новый файл *.snk (все возможные команды можно вывести с помощью опции -?):

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

Можно указать компилятору C#, чтобы он использовал эти ключевые данные для назначения строгого имени библиотеке CorLibDumper.dll, добавив в файл LibraryBuild.rsp следующие команды:

```
/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 следует ознакомиться с утилитой командной строки monodis, которая функционально эквивалентна утилите ildasm.exe от Microsoft (но без графического интерфейса). Утилита monodis позволяет просматривать CIL-код, манифест и метаданные типа для указанной сборки. В нашем случае необходимо просмотреть основную информацию о (уже строго именованной) сборке с помощью флага --assembly.

Результат выполнения приведенной ниже команды показан на рис. Б.5:

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

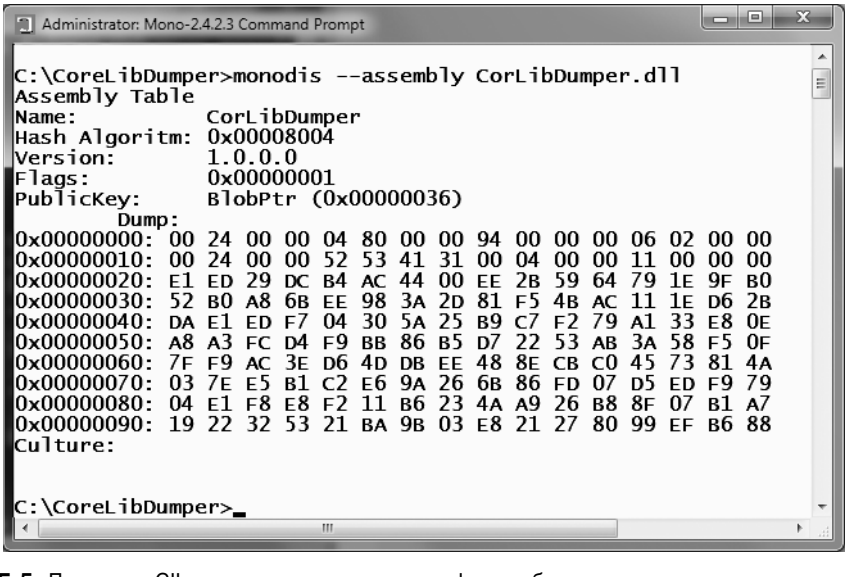

Рис. Б.5. Просмотр CIL-кода, метаданных и манифеста сборки с помощью утилиты monodis

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

Установка сборок в Mono GAC

Итак, вы уже снабдили сборку `CorLibDumper.dll` строгим именем, и теперь ее можно установить в Mono GAC. Как и одноименное средство от Microsoft, утилита `gacutil` из состава Mono может устанавливать и удалять сборки, а также выводить список сборок, установленных в кеше `C:\Program Files\Mono-<версия>\lib\mono\gac`. Следующая команда развертывает сборку `CorLibDumper.dll` в GAC и регистрирует ее в компьютере как открытую сборку:

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

На заметку! Для установки этого двоичного файла в Mono GAC обязательно применяйте командную строку Mono! Если воспользоваться разработанной Microsoft программой `gacutil.exe`, то сборка `CorLibDumper.dll` будет установлена в Microsoft GAC.

Если после выполнения этой команды открыть каталог `\gac`, то вы увидите там новую папку `CorLibDumper` (рис. Б.6).

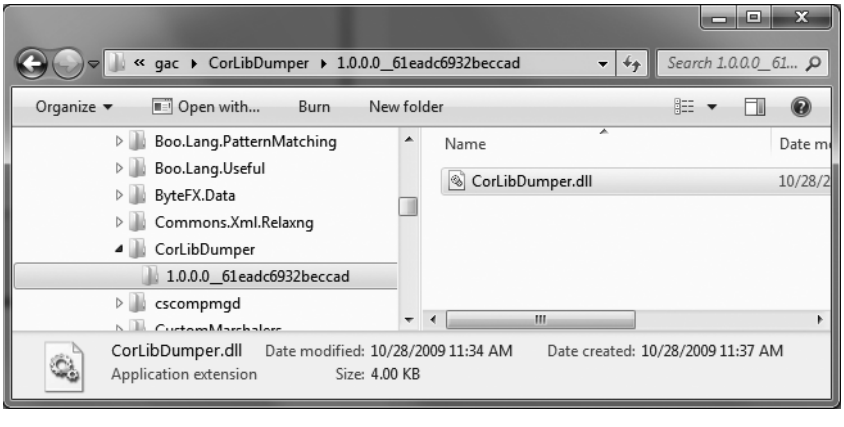

Рис. Б.6. Развертывание библиотеки кода в Mono GAC

Эта папка определяет подкаталог, который следует тем же соглашениям по именованию, что и Microsoft GAC (*версияСборки__маркерОткрытогоКлюча*).

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

Создание 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`, который содержит следующее определение 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()` использует введенное пользователем имя для вывода членов типа в локальный файл. Теперь создайте для этого приложения файл `ClientBuild.rsp` и укажите ссылку на `CorLibDumper.dll`:

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

Воспользуйтесь командной строкой Mono для компиляции исполняемой сборки с помощью `mcs`:

```
mcs @ClientBuild.rsp
```

Загрузка клиентского приложения в среду выполнения Mono

После этого приложение `ConsoleClientApp.exe` можно загрузить в механизм времени выполнения Mono, указав в качестве аргумента имя исполняемого файла (с расширением `*.exe`):

```
mono ConsoleClientApp.exe
```

Для проверки введите в окне командной строки `System.Threading.Thread` и нажмите клавишу <Enter>. Вы увидите новый файл с именем `System.Threading.Thread.txt`, который содержит определение метаданных типа (рис. Б.7).

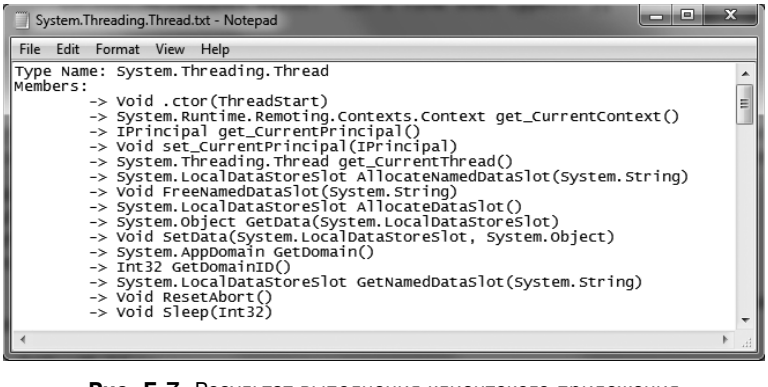

Рис. Б.7. Результат выполнения клиентского приложения

Прежде чем перейти к созданию клиента Windows Forms, попробуйте провести следующий эксперимент. С помощью проводника Windows переименуйте сборку `CorLibDumper.dll` в папке, содержащей клиентское приложение, на `DontUseCorLibDumper.dll`. Клиентское приложение все равно должно успешно запуститься, т.к. доступ к данной сборке при создании клиента необходим лишь для изменения манифеста клиента. Во время выполнения среда выполнения Mono загрузит версию `CorLibDumper.dll`, развернутую в Mono GAC.

Но если вы попытаетесь выполнить `ConsoleClientApp.exe`, дважды щелкнув на нем в проводнике Windows, то увидите, что сгенерировано исключение `FileNotFoundException`. На первый взгляд может показаться, что оно возникло из-за переименования сборки `CorLibDumper.dll` в папке клиентского приложения. Однако настоящей причиной является то, что вы загрузили `ConsoleClientApp.exe` в Microsoft CLR!

Чтобы запустить приложение в среде Mono, необходимо передать его механизму выполнения Mono с помощью команды `Mono`. Иначе сборка будет загружена в Microsoft CLR в предположении, что все открытые сборки, установленные в Microsoft GAC, находятся в каталоге `<%windir%>\Assembly`.

На заметку! Двойной щелчок на исполняемом файле в проводнике Windows загружает эту сборку в Microsoft CLR, а не в механизм выполнения Mono!

Создание клиентской программы Windows Forms

Теперь переименуйте файл `DontUseCorLibDumper.dll` снова на `CorLibDumper.dll`, иначе следующий пример не сможет скомпилироваться. Затем создайте новый файл кода C# по имени `WinFormsClientApp.cs` и сохраните его в той же папке, что и текущие файлы проекта. В этом файле определены два класса:

```
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);

      // Обработать анонимно 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);
    }
  }
}
```

Для компиляции этого приложения Windows Forms с помощью файла ответов создайте файл с именем `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
```

Теперь сохраните этот файл (обязательно в той же папке, что и код примера) и передайте его в качестве аргумента компилятору `mcs` (с помощью элемента `@`):

```
msc @WinFormsClientApp.rsp
```

И, наконец, запустите полученное приложение Windows Forms с помощью Mono:

```
mono WinFormsClientApp.exe
```

Возможный результат тестового запуска показан на рис. Б.8.

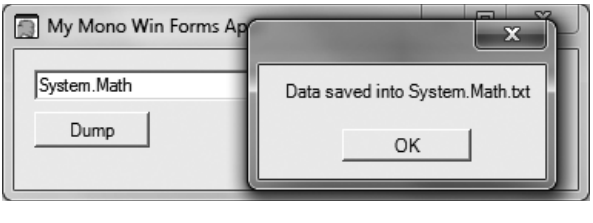

Рис. Б.8. Приложение Windows Forms, построенное с использованием Mono

Выполнение приложения Windows Forms под Linux

До сих пор в этом приложении демонстрировалось создание сборок, которые можно было бы скомпилировать и с помощью Microsoft .NET Framework 4.0 SDK. Однако важность Mono станет понятнее, если вы взглянете на рис. Б.9, где показано то же самое приложение Windows Forms, работающее под SuSe Linux. Приложение Windows Forms имеет примерно тот же внешний вид (с учетом текущей темы рабочего стола).

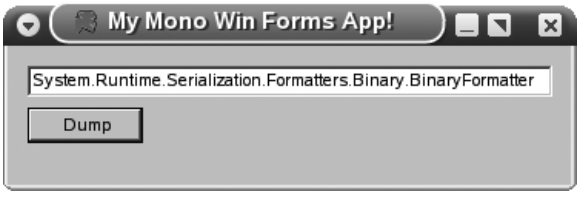

Рис. Б.9. Выполнение приложения Windows Forms под SuSe Linux

Исходный код. Проект `CorLibDumper` находится в подкаталоге `Appendix B`.

Оказывается, в точности тот же самый код C#, приведенный выше в приложении, можно скомпилировать и запустить под Linux (а также под управлением любой другой операционной системы, поддерживаемой Mono) с помощью тех же средств разработки Mono. Любую сборку, созданную без использования программных конструкций .NET 4.0, можно развернуть или перекомпилировать для другой операционной системой, совместимой с Mono, а затем выполнить с помощью утилиты времени выполнения `Mono`. А поскольку все сборки содержат код CIL, не зависящий от платформы, приложения вообще не нужно перекомпилировать.

Кто использует Mono?

В этом кратком приложении платформа Mono была обрисована с помощью нескольких простых примеров. Если вы собираетесь создавать программы .NET только для семейства операционных систем Windows, то можете и не повстречаться с компаниями или отдельными лицами, которые активно используют Mono. Но все-таки Mono живет и процветает в сообществах программистов для Mac OS X, Linux и Windows.

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

```
http://mono-project.com/Software
```

Там находится длинный список коммерческих продуктов, созданных с помощью Mono, включая средства разработки, серверные продукты, видеоигры (в том числе игры для Nintendo Wii и iPhone) и медицинские прикладные системы.

Если вы не пожалеете времени и пройдете по ссылкам, то увидите, что Mono отлично оснащена для создания действительно межплатформенного программного обеспечения .NET, которое реально работает вплоть до уровня предприятия.

Рекомендации по дальнейшему изучению

Если вы внимательно прочитали весь материал, изложенный в книге, то уже много знаете о Mono, т.к. это реализация CLI, совместимая с ECMA. Если вы хотите узнать больше об особенностях Mono, то лучшим местом для начала будет официальный веб-сайт Mono (`www.mono-project.com`). Там имеется страница `www.mono-project.com/Use`,

которая может служить отправной точкой для ряда важных тем, в числе которых доступ к базам данных с помощью ADO.NET, веб-разработка с помощью 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" (Среды разработки Mono: выход за пределы командной строки) — в этой статье рассматриваются многие IDE-среды, пригодные для разработки приложений Mono.
- "Building Robust UIs in Mono with Gtk#" (Создание надежных пользовательских интерфейсов в Mono с помощью Gtk#) — здесь описывается создание графических настольных приложений с помощью инструментального набора GTK# в качестве альтернативы Windows Forms.

И, наконец, рекомендуется ознакомиться с веб-сайтом документации по Mono (`www.go-mono.com/docs`). Здесь вы найдете документацию по библиотекам базовых классов Mono, средствам разработки и другим темам (рис. Б.10).

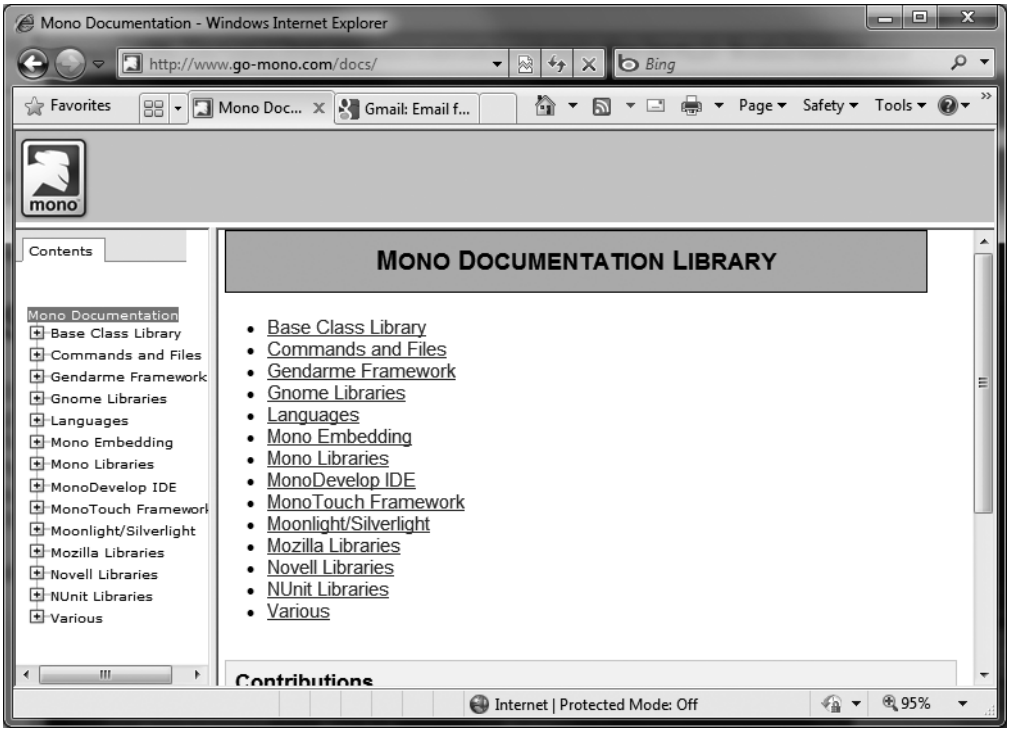

Рис. Б.10. Онлайновая документация по Mono

На заметку! Веб-сайт с онлайновой документацией по Mono поддерживается сообществом разработчиков. Поэтому не удивляйтесь, если обнаружите ссылки на незавершенные документы! А поскольку Mono является дистрибутивом Microsoft .NET, совместимым с ECMA, то при изучении Mono можно воспользоваться онлайновой документацией MSDN.

Резюме

Данное приложение задумано как введение в межплатформенную природу языка программирования C# и платформы .NET при использовании инфраструктуры Mono. Вы ознакомились с рядом инструментов командной строки, входящих в состав Mono, которые позволяют создавать различные сборки .NET, в том числе и строго именованные сборки, разворачиваемые в GAC, приложения Windows Forms и библиотеки кода .NET. В любом случае, если вам понадобится создавать приложения .NET, которые должны выполняться под различными операционными системами, то проект Mono будет великолепным вариантом.