

Practical Mono



Mark Mamone

Practical Mono

Copyright © 2006 by Mark Mamone

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-548-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jason Gilmore

Technical Reviewer: Marcus Fletcher

Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser

Project Manager: Pat Christenson

Copy Edit Manager: Nicole LeClerc

Copy Editors: Kim Wimpsett, Ami Knox

Assistant Production Director: Kari Brooks-Copony

Compositor: Susan Glinert

Proofreaders: Lori Bring, Kim Burton, Linda Seifert, Dan Shaw, Nancy Sixsmith

Production Editing Assistant: Kelly Gunther

Indexer: Ann Rogers

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Director: Tom Debolski

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit <http://www.springeronline.com>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Source Code section.



Introducing .NET and Mono

Ever since the .NET initiative launched in January 2000 and the Visual Studio .NET beta subsequently launched in October 2000, the .NET train has been gaining speed; it now is the number-one development platform for the Windows operating system. However, you may have noticed a key point in that statement: for the Windows operating system! Yes, you guessed it: Microsoft targeted the .NET technology suite at the Windows operating system only; therefore, at the time of its launch, .NET was not available on other platforms such as Unix, Mac OS X, or Linux. This is not surprising, as Microsoft develops the Windows operating system and doesn't want to invest time in creating versions for competing operating systems; however, in the corporate world, other operating systems exist and for good reasons.

In this chapter, you'll learn exactly what .NET is and what it consists of; it's important to note that Microsoft made a key decision that acted as the catalyst for the Mono project and ultimately enabled .NET to run on operating systems other than Windows. This decision was to submit the specification for a core component of the common language runtime (CLR) and the Common Language Infrastructure (CLI) to the international standards body ECMA International, an industry association "dedicated to the standardization of information and communication systems." What does this mean? It means that anybody can obtain the specification for core .NET components and in doing so develop their own implementation of that specification, effectively writing their own .NET Framework components.

In addition, this chapter will introduce Mono and its associated elements including its tools and configuration. You'll also take a brief look at integrated development environments (IDEs), which are graphical applications that make developing your application as little easier. Finally, this chapter will cover what you need to know about the sample application you'll develop throughout this book, which is a fully functional RSS aggregator.

Microsoft Introduces .NET

In 2000, use of the Microsoft Windows operating system was widespread on desktop and laptop PCs, both at home and at work. In addition to the numerous other desktop products (such as Microsoft Office) and server products (such as SQL Server, Exchange, and so on) Microsoft produces, it had achieved success with development technologies such as C++, Visual Basic, Active Server Pages (ASP), the Common Object Model (COM), and so on. However, Java was starting to gain steam, and Microsoft needed to evolve its technology stack to help combat this threat and others in the open-source community (such as Python and PHP). After hiring the lead designer of Borland Delphi, Microsoft announced .NET 1.0 in the summer of 2000 for

a 2001 launch (see <http://www.pcworld.com/news/article/0,aid,17397,00.asp>). Thereafter, Microsoft launched version 1.1 of the .NET Framework in April 2003, and more recently, it announced version 2.0, which is to be launched late in 2005.

What Is .NET?

Microsoft .NET is, according to Microsoft, “a strategy for connecting systems, information, and devices through Web Services.” This is true; however, it doesn’t do justice to the power of the .NET Framework, so I’ll describe it in a slightly different way.

.NET is a software platform that provides several technologies that benefit corporate organizations, individuals, and the development community. Its aim is to provide a set of technologies that will offer ubiquitous interoperability through the standards it embraces and defines. At a high level, .NET consists of several elements, such as the .NET Framework and class library. I’ll introduce each of these in the following sections.

The .NET Framework

As its name implies, the .NET Framework is an all-encompassing framework or environment for constructing, deploying, and running applications that leverage the features of .NET, including Extensible Markup Language (XML), Web Services, Windows Forms, ADO.NET, and so on. In addition, the .NET Framework also encompasses a companion framework, called the .NET Compact Framework, that is targeted at mobile devices or indeed any device that can run the CLR, including set-top boxes.

Common Language Runtime (CLR)

The CLR is the foundation that underpins all .NET applications; it’s the engine that drives the vehicle. The CLR is an execution environment (hence the term *runtime*) that supports the execution of applications written in Intermediate Language (IL), the product of compiling your .NET application. The CLR is standards based; therefore, it is possible for you to develop your own version for your favorite operating system, which is exactly what has happened with the Mono environment (although it is no simple task!). The CLR offers a *managed* environment that supports a number of advanced features, including the following:

- Multiple library versions
- Just-in-time (JIT) compilation
- Automatic memory management

It would be easy to assume what some of these features offer without fully understanding their benefits, so I’ll provide a brief overview of what benefit these advanced features offer in the following sections.

Multiple Library Versions

A well-known problem on the Windows platform, and to a lesser extent on other operating systems, is the complexities associated with deploying multiple versions of the same components. This problem manifests itself by stopping working applications from running correctly,

because a new version of a component has been installed on which the application relies. The .NET Framework solves this by allowing different versions of the same component to be installed side by side; applications simply continue using the version of component against which they were built. These libraries in the .NET world are also known as *assemblies*, which are something I cover in Chapter 5.

Just-in-Time Compilation

One trait of the managed environment that the .NET Framework offers is the compilation of code into an intermediate language that is then executed by the CLR. This compilation happens *just-in-time*, which means the code is compiled when first referenced and thereafter the already compiled code can be executed. This offers the benefits of an interpreted language whilst reducing the performance impact.

Automatic Memory Management

One of the features of a managed environment is the automatic memory management that the CLR provides. This memory management allocates and releases memory automatically without your application's code having to explicitly do it—although some best practices (discussed in Chapter 12 of this book) can help the CLR perform to the best of its ability.

Note This memory management is associated only with *managed code*, that is, code running under the control of the CLR. The .NET Framework allows you to still write and execute code written in unmanaged languages such as C++, which does not run under the control of the CLR but can interact with managed code through features within the .NET Framework.

Class Library

The CLR alone would not constitute an easy development environment in which to build. You'd need to know the complexities of IL, and you'd be going back to the days of assembler programming. This is obviously still something you can do, should you want to harness all the power possible at the lowest level; however, it's time-consuming and largely unnecessary.

The class library that accompanies the .NET Framework is a set of comprehensive classes, interfaces, and value types that provide access to the system's functionality and services. They are the building blocks on which you create .NET applications, components, and custom controls.

.NET As an Open Industry Standard

Microsoft made an important decision early on in the development of the .NET Framework: to submit the specifications for key components of the .NET Framework to the international standards organization ECMA International. The result is that the specifications are now standard and available to anyone who wants to implement their own version of the .NET Framework. You can view the final standards on the ECMA International Web site at <http://www.ecma-international.org>.

Command Language Infrastructure (CLI)

The CLI is the overarching specification (ECMA 335) that allows you to write an application in a high-level programming language and execute it on any platform that provides an implementation of the CLR. The CLI consists of the following:

- A file format
- A common type system (CTS)
- An extensible metadata system
- IL code
- Access to the native operating system services
- A base class library

For clarity, in the following sections, I'll provide a brief overview of what some of these key components of the CLI actually mean.

Common Type System (CTS)

The CTS provides a number of guiding principles to enable the .NET Framework to support multiple but interoperable languages. It provides a set of rules that a .NET language must follow in order to interact with other languages; these include how to handle types (both *reference* and *value*) and how to provide a model for object orientation. This is a key feature of the .NET Framework and means that you could write your application using one or more of the languages supported by the .NET Framework, including C#, Visual Basic .NET (VB .NET), Python, and more.

For a list of the languages supported (although it's not exhaustive), see <http://www.dotnetlanguages.net/DNL/Resources.aspx>.

Note Your application could have different parts written in different languages. These could be one of the managed languages available such as C# or an unmanaged language such as C++, which you could then integrate.

Intermediate Language (IL)

Once you have written your application in one (or more) of the languages mentioned, you then compile the application (using the JIT compiler) into an intermediate language known as IL. The CLR then executes the IL. This language is platform independent though its definition is a standard file format that an implementation of the CLR can understand. It's this file format that offers the ability for you to run any .NET code on any platform, provided it has an implementation of the CLR available.

Note You can even write your applications directly in IL if you want!

To summarize, the file format provides a standard for representing files, the CTS provides the interoperability of languages and applications, IL provides platform interoperability and allows further compilation in native systems, and finally the metadata system and base class library complete the features necessary for component-based development.

Other Technology Support, Such As Extensible Markup Language (XML)

In addition to defining standards for adoption and supporting interoperability with other technologies and platforms, the .NET Framework embraces other well-known standards such as XML and Web Services (the provision of functionality over the common HTTP channel).

While this does not guarantee interoperability, it promotes it, and provided that both parties in communication observe the standards present, code can now happily coexist and talk regardless of the language it was written in (in other words, Java, C#, and so on), the runtime it executes under (that is, the Java Runtime Environment [JRE] or CLR), and the platform it runs on (that is, Windows, Linux, and so on).

Technologies

In addition to the foundation provided by the CLR and all its components, .NET provides you with several other technologies and services for you to embrace within your applications. The following sections describe some of these technologies.

Active Server Pages .NET (ASP.NET)

The ASP technologies implemented by previous versions of Windows and Internet Information Services (IIS) received wide adoption by the information technology (IT) industry, and Microsoft rearchitected the concept of server-side, dynamic scripting code when it introduced ASP.NET. This allows you to implement your Web site using all the power and features of the .NET Framework and class libraries when developing powerful, server-side browser-based applications and/or Web sites. Another key architectural feature is the implementation of code-behind pages, where the graphical user interface (representing the content) and the back-end code (representing the business logic) is separate.

ADO.NET

The implementation of Active Data Objects (ADO) for .NET is an evolutionary step in providing a common framework for accessing disparate sources of data. It provides four key advantages:

- *Interoperability:* You gain interoperability, because the data format underpinning ADO.NET is XML, an industry standard understood by most platforms.

- *Scalability*: You achieve scalability through the support of advanced features such as disconnected datasets and connection pooling.
- *Productivity*: You gain productivity through the adoption of standard data components that use abstraction to connect to any data source but that are strongly typed for improved reliability.
- *Performance*: You achieve productivity through the support of features that also benefit scalability. Connection pooling reduces the overhead of establishing connections; data classes are provided to inherit the native speed of database libraries or the interoperability of concepts such as Open Database Connectivity (ODBC).

Safe, timely, and easy access to data is critical to the success of a framework, and the power of ADO.NET makes these tasks easy without sacrificing other important factors such as speed.

.NET Tools

The .NET tools are necessary to create, execute, and debug .NET applications; they include the categories of tools described in the following sections.

Command-Line Tools

Providing command-line based tools as part of the .NET Framework allows developers to start developing .NET applications straightaway. You don't need to purchase expensive commercial, off-the-shelf (COTS) packages, although you can if you want some of the advanced functionality these typically offer. The .NET Framework ships with a compiler, runtime environment, assembler, disassembler, class viewer, and much more. I'll cover these in more detail in Chapter 2.

Commercial, Off-the-Shelf (COTS) Packages

In addition to the command-line tools and open-source tools such as MonoDevelop, you also have the option to purchase (or write!) COTS packages that often provide functionality that is more advanced than that available in the open-source community (although this is becoming less true; take a look at MonoDevelop!). The most obvious candidate for this is Microsoft Visual Studio, an extremely powerful development environment that runs on the Windows platform only. It is on this product that the MonoDevelop tool is modeled (although it was born as the SharpDevelop tool). Another IDE is Eclipse (see <http://www.eclipse.org/>), which can provide support for the .NET Framework and C# through a downloadable plug-in (see <http://www.improve-technologies.com/alpha/esharp>).

The COTS products in the Windows world are big business. In the open-source community, though, the driver is "sharing" rather than "the bottom line," so fewer COTS products are sold. However, commercial products are sold that are built upon Mono and other open-source technologies; Novell's iFolder is a good example.

The History of Mono

Launched in mid-2001 by Ximian cofounder Miguel de Icaza, Mono is an open-source initiative that brings .NET technologies to operating systems other than Microsoft Windows. At the time of this writing, the Mono project has .NET distributions available for Linux, Windows, Mac OS, Berkeley Software Distribution (BSD), and Solaris operating systems. It also includes support for the x86 (32-bit and 64-bit) and PowerPC (PPC) processors. In doing so, this allows you to use Mono to develop cross-platform applications simply and easily.

Icaza started the GNU Object Model Environment (GNOME) project in August 1997 with a friend, Federico Mena. Although the project was successful, Icaza was becoming frustrated with the complexity of developing component-based solutions and liked Microsoft's .NET platform. It was a "good platform to innovate from."¹ He also wanted to promote the concepts introduced as part of the Bonobo project (see <http://developer.gnome.org/arch/component/bonobo.html>), a Microsoft COM-influenced component model but on the .NET platform. Hence, the Mono project was born.

Icaza cofounded Ximian with the aim of producing and marketing Linux-based productivity software. GNOME featured at the core of this software suite, and it included additional value-add applications such as Evolution (<http://www.novell.com/products/evolution>), a comprehensive calendar/e-mail client and contact management application.

Icaza is now the vice president of development at Novell (<http://www.novell.com>); Novell acquired the Ximian group in 2003, so Mono is now a fundamental part of the Novell strategy, with a number of its core applications and services, such as Novell Linux Desktop 9, being based on the Mono development environment.

What Is Mono?

So, you should now have an idea of what .NET is. If not, don't worry too much, but a key message to take away from the previous sections is that the .NET Framework, as written by Microsoft, is available for the Windows platform only. But people often ask the question, Is .NET available for the Unix operating system? What about Apple's Mac OS? What about Linux? The answer to all of these questions was "no!"—until the advent of Mono, that is.

Mono is an open-source (community-built) implementation of the .NET Framework and associated components for platforms other than Windows. The Mono environment can be classified as the "core Mono" environment and "additional components" that offer enhanced functionality that will often be built upon the core Mono environment. Let's look at what is typically contained within these elements.

Core Mono

The core Mono environment is where you should start; it provides key components that are required to create and host .NET applications. Chapter 5 covers the key topics introduced in the following sections.

1. From "GNOME's Miguel de Icaza on .NET: The New Development Environment for the Next 20 Years" at <http://www.ondotnet.com/pub/a/dotnet/2001/07/09/icaza.html>

Common Language Runtime (CLR) Implementation

This provides an implementation of the CLR through an executable that invokes your .NET application within the bounds of the CLR.

C# Compiler

A core part of the Mono development environment, the C# compiler comes standard with the option of downloading compatibility options for both Java and Visual Basic, although the compilers are still in development. For example, at the time of writing, MonoBasic (mbas) is still heavily in development, although I recommend keeping an eye on the Mono Web site for the latest details. For example, check out http://mono-project.com/Visual_Basic for the latest Visual Basic news.

.NET Class Library Implementation

The Mono environment also includes a set of libraries that constitute the .NET class library implementation, which is the Mono implementation of the .NET class library explained previously. As you can imagine, this is a big task, and you can check the status of the various classes available at <http://www.go-mono.com/class-library.html>. Of course, the nature of the open-source environment means there's always work to do and contributions are always welcome.

GNOME, Mono, or Unix Libraries

In addition to the .NET implementations provided through Mono, the fact that Mono lives on platforms other than Microsoft means that embracing other technologies is a natural enhancement. One such example is the provision of a GNOME toolkit called Gtk#, which provides a set of C# bindings that can be used to develop GNOME-based applications.

Development Tools

In addition to some of the tools mentioned, other tools are provided that are covered in Chapter 2. Some examples include assemblers and disassemblers for IL, digital certification utilities, assembly cache utilities, and more.

Additional Mono Components

In addition to the core Mono components, Mono 1.0 includes support for other components. Figure 1-1 shows a high-level view of the Mono components, and I should stress that this functionality is being extended on a daily basis.

The following sections describe some of the components, and I'll cover the other elements in more detail in Chapter 5. For now it's important only that you understand the concepts.

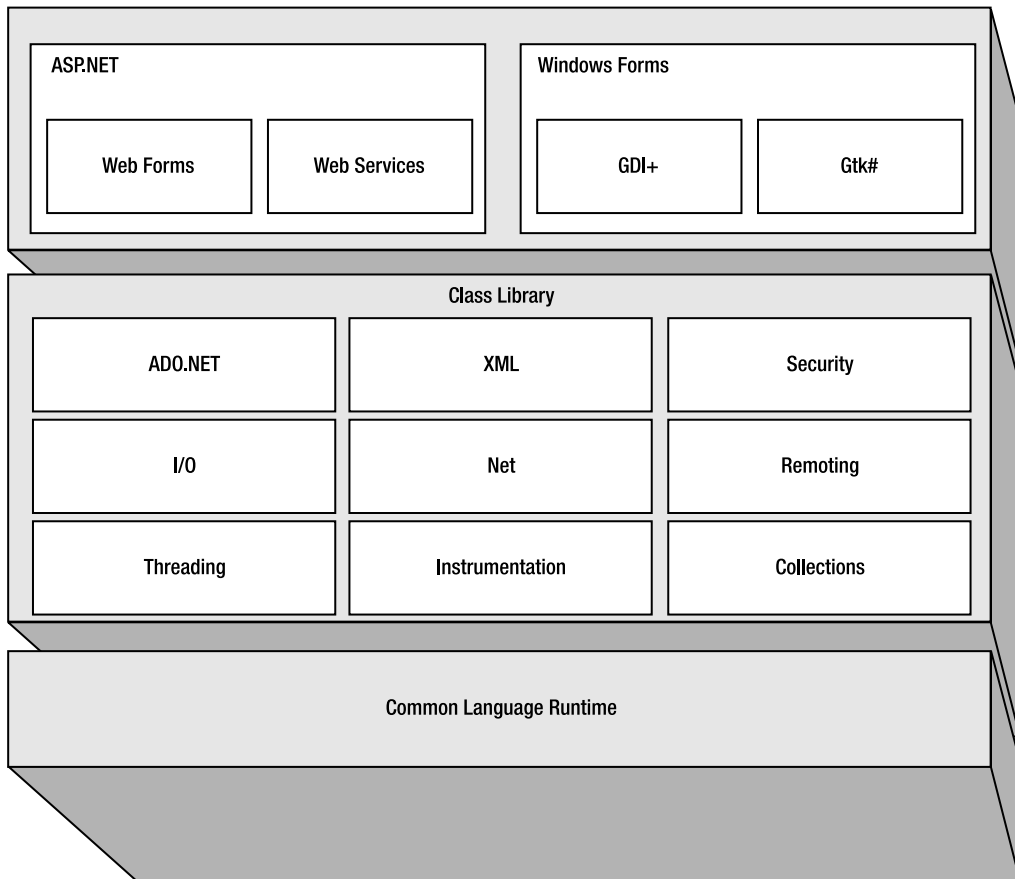


Figure 1-1. *High-level view of the Mono components*

ASP.NET Web Forms

The ability to create dynamic server-side Internet applications that provide the rich functionality normally associated with rich client applications is the forte of the ASP.NET Web Forms technology. It allows you to design and construct Web pages (or *Web Forms*, as they are termed) using a graphical user interface (GUI) designer that are then transformed into a Hypertext Markup Language (HTML)-based implementation that can be viewed through a standard Web browser. This component provides the environment and functionality to interpret the Web Forms and allows you to provide business logic in a .NET-compliant language such as C#. Chapter 11 describes how to harness some of the functionality within ASP.NET.

ASP.NET Web Services

In addition to providing support for Web Forms, the ASP.NET component also provides support for Web Services. This offers the ability for you to expose business functionality using standard protocols that can then be consumed by other systems across the Internet. For example, you may want to write a Web Service that offers the ability to look up the latest stock value. I'll cover the power of Web Services in Chapter 11.

ADO.NET

At some point, you'll need to access a database to provide some persistent storage for your application. A typical example is when your solution needs to take advantage of the transactional capabilities of a relational database management system (RDBMS), committing or rolling back transactions as required. A data abstraction layer, in the form of ADO.NET, simplifies access to database functionality but still retains the power you will typically require. You'll see what ADO.NET is and how to use it in Chapter 8.

Remoting

This is a fairly complex topic, but imagine the ability for applications to communicate using Remote Procedure Call (RPC) technologies, that is, for your application to call the method of an object as if it were local to your machine, when in fact it could be halfway across the world! I'll cover this in more detail in Chapter 10.

XML and Serialization

.NET embraces XML at its core, and it's probably no surprise that its adoption and its support within the .NET environment are comprehensive. XML is ubiquitous within not only the world of the Internet but also the commercial world. The .NET class libraries provide comprehensive support for XML manipulation; I'll demonstrate how to use them in Chapter 9.

In addition to the aforementioned elements, .NET provides many other cool features with powerful capabilities, including Web Services, language neutrality, mobile development, and much more.

How to Get Involved with Mono

As Mono is part of the open-source community, your contributions are always welcome, and how you choose to contribute is pretty much up to you. The community is coordinated via the Mono home page (<http://www.mono-project.com/>); for example, you could fix a bug or complete an unfinished class from a list. In either case, your source code is then booked back into source code control and integrated into the Mono code base through controlled build and testing procedures.

How to Obtain Mono

You've been introduced to Microsoft .NET, and you've learned the story of how Mono started and what Mono is. Now you'll learn how you obtain the software, install it, and start using the powerful features I'll explain within this book.

One method is to ensure that you have a suitable host operating system on which to run. A native installation such as this involves installing the software onto a supported platform such as one of the approved Linux distributions, and—hey, presto!—you're off and running.

Another method, and one that is ideal if you want to start to learn Mono with the minimum amount of impact to your desktop, is to use Monoppix, which makes Mono available on a distribution of Linux that is based on Knoppix and that boots and runs entirely from CD. You can find it at <http://www.monoppix.com/>.

However, for the purpose of this book, I'll show native installations. So, the first step is to ensure that you download the components appropriate to the platform you are running. At the time of writing, these are split into those considered stable (older but tested versions) and those considered unstable (newer but not fully tested):

- Linux
- Microsoft Windows 2000 and above
- Mac OS X

For Linux, the following x86 processor distributions are recognized:

- Red Hat 9.0 and Fedora Core 3.0
- SUSE Linux Enterprise Server 9, SUSE Linux 9.0, and SUSE Linux 9.1 (see <http://www.novell.com/linux/suse/>)

Some of the “unstable” releases include a version for the Mac OS X operating system and 64-bit processor implementations. The source code is also available as a separate download from the packages discussed next.

You can find all these versions on the Mono home page under Downloads (<http://www.mono-project.com/Downloads>). I'll show how to implement the solution in this book on the Fedora Core 3 distribution, and you can download the packages for this as appropriate. Figure 1-2 shows the downloads section for Fedora Core 3.

Note The Microsoft Windows download is slightly different in that the download offered is a single installation module that includes Gtk# and the XSP Web server.

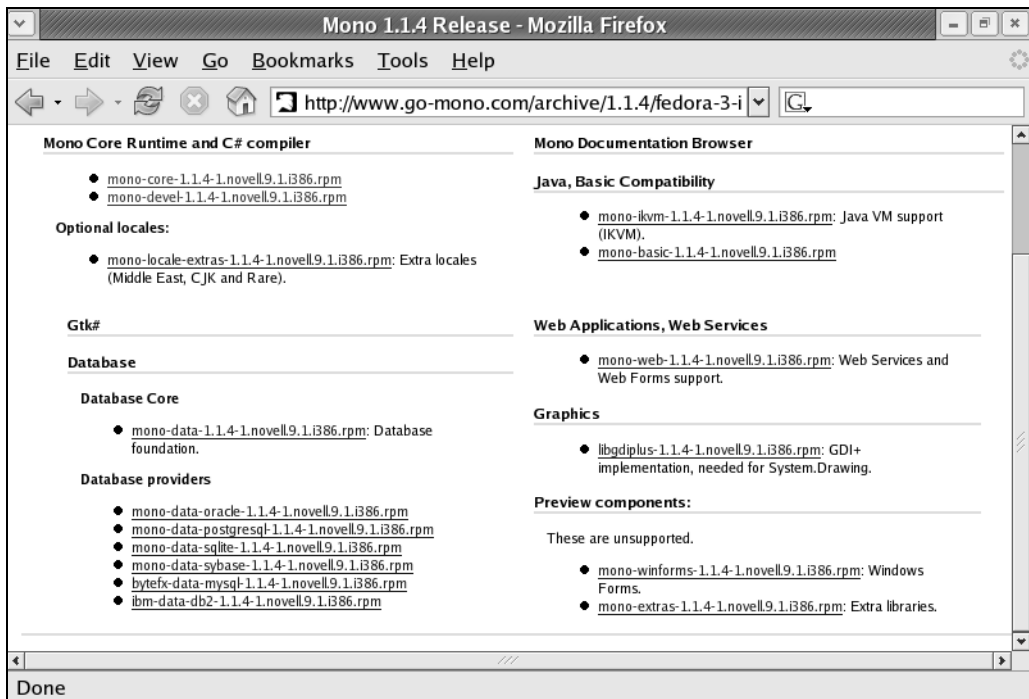


Figure 1-2. Sample archive page at the Mono home page

The next step is to download the required components, which may be listed either individually or grouped; in either case, you won't need all the packages, but I'll briefly explain what some of the components do before continuing:

- *Mono core runtime and C# compiler:* This provides the CLI implementation and a C# compiler, along with some other core command-line tools such as an assembler, disassembler, and global assembly cache (GAC) utility.
- *Mono development tools:* As fun as the command-line may be, the Mono development tools provide a comprehensive IDE, similar to that provided by Microsoft Visual Studio.
- *Gtk#:* This provides the necessary libraries to write applications that utilize the GNOME toolkit.
- *Java, basic compatibility:* These packages provide support for other languages and basic compatibility; for example, if you're more comfortable using the Java language, this is the package to download.
- *Web applications and Web Services:* This provides support for Web-based applications and Web Services, including ASP.NET technologies that are contained within these packages.
- *Database core:* This provides the ADO.NET implementation, allowing your .NET applications to connect to external data sources using the appropriate data provider.

- *Database providers*: These data providers provide access to well-known external data sources such as industry-standard relational databases. An example of a data provider is the MySQL data provider.
- *Graphics*: These packages provide the implementation of GDI+, the .NET graphics layer, allowing you to write complex graphical applications.
- *Preview components*: This package provides components that are currently undergoing development and so are considered “preview” components. It can be fun to experiment with the up-and-coming components found in these packages.

How to Install Mono on Linux

To get started, I’ll ask you to download the following packages for now, and then you can download further packages as I introduce the topics that require them throughout this book. So, to start, download the Mono Core Runtime, Mono Development Tools, and C# Compiler packages.

You can install Mono using other methods (for example, using the Novell Red Carpet FTP service, compiling from source code, and so on), but I’ll focus on the simplest, albeit more laborious, method.

Note If you are new to Linux, Red Hat Package Manager (RPM) and is a standard method for deploying software on the Linux platform using the RPM Package Manager module. This is usually used for installing and removing Linux software packages (although it’s not the only option by any means).

Once you’ve downloaded your files, you need to install them. You’ll need to ensure that you’re logged in as the root user and then can install all the RPM packages in the current directory by using the following command line:

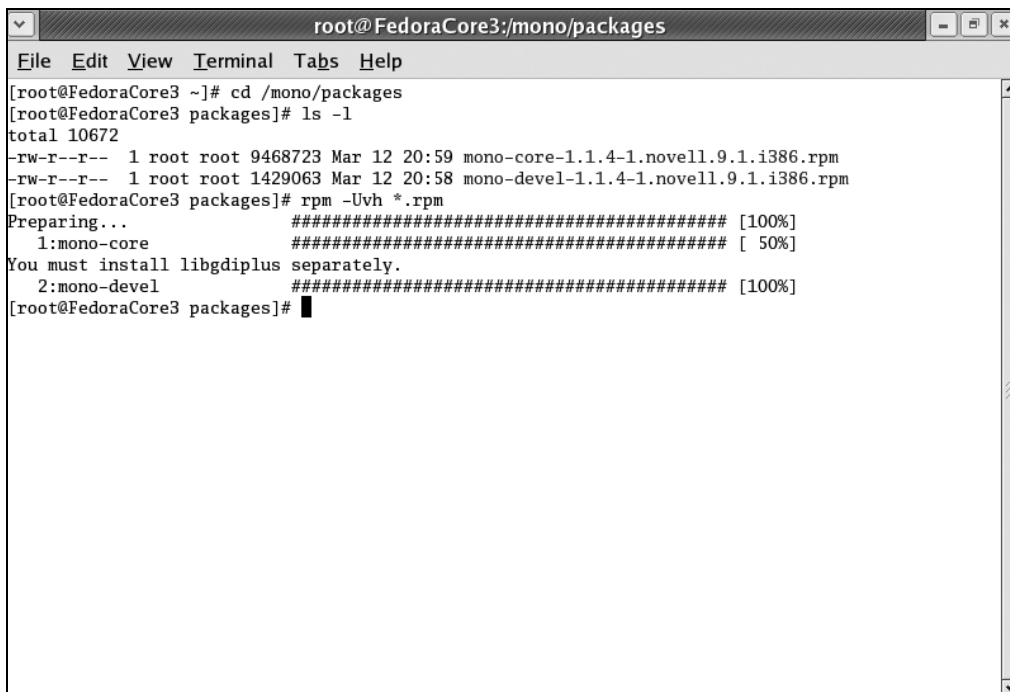
```
rpm -Uvh *.rpm
```

Or, you can install them individually, if you want to be more specific, using the following command line:

```
rpm -i mono-core-*.rpm  
rpm -i mono-devel-*.rpm
```

Tip The filename downloaded will reflect the version number, so to avoid confusion I’ve used an asterisk. However, you must ensure that you either have only one version present before installation or explicitly reference the filename.

I chose the former option, which should install the packages successfully. Figure 1-3 shows how your screen should look.



```

root@FedoraCore3:/mono/packages
File Edit View Terminal Tabs Help
[root@FedoraCore3 ~]# cd /mono/packages
[root@FedoraCore3 packages]# ls -l
total 10672
-rw-r--r-- 1 root root 9468723 Mar 12 20:59 mono-core-1.1.4-1.novell.9.1.i386.rpm
-rw-r--r-- 1 root root 1429063 Mar 12 20:58 mono-devel-1.1.4-1.novell.9.1.i386.rpm
[root@FedoraCore3 packages]# rpm -Uvh *.rpm
Preparing... ##### [100%]
 1:mono-core ##### [ 50%]
You must install libgdiplus separately.
 2:mono-devel ##### [100%]
[root@FedoraCore3 packages]#

```

Figure 1-3. *Linux core Mono installation*

Tip You may find you're missing some dependent files. For example, a vanilla Fedora Core 3 installation will be missing some files that you can find within the `libc_u-3.2-1.i386.rpm` file; this is provided with the Fedora Extras folder on the Fedora site. Another good source of missing RPMs is the RPMforge site at <http://www.rpmforge.net>.

You can check whether the installation was successful by executing the following command line:

```
mono --version
```

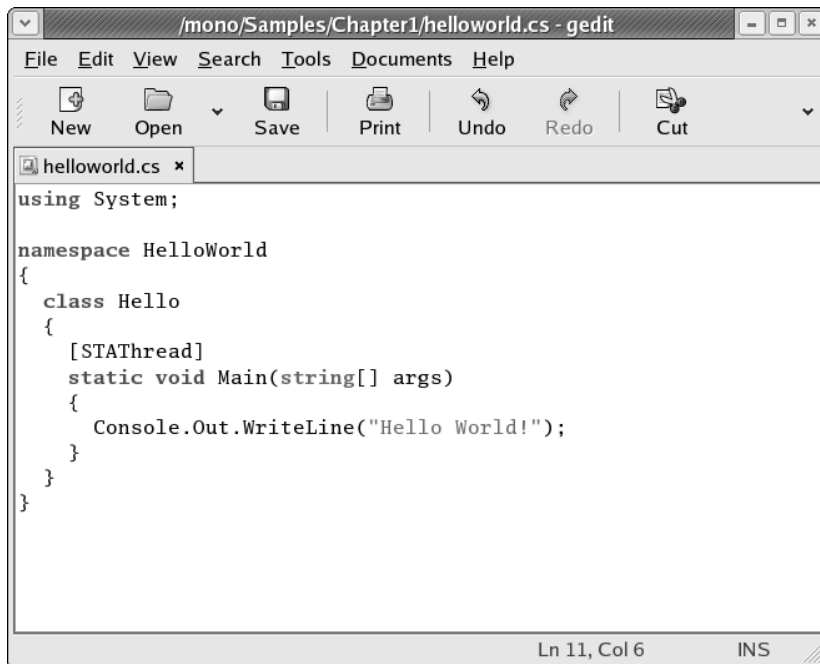
This should report the version of Mono that was installed; in addition, you can double-check by looking for some key tools that are provided as part of the Mono command-line utilities in the `/usr` folder (see Table 1-1). I'll briefly categorize these, but I'll discuss these and more comprehensive tools throughout the book as appropriate and primarily within Chapter 2.

Table 1-1. *Common Command-Line Utilities*

Executable Name	Description
monodis	Disassembler
ilasm	Assembler
mcs	C# compiler
mono	CLI runtime implementation
gacutil	GAC utility

Tip Once you have finished installing Mono as the root user, I recommend switching to your standard user profile to avoid problems through mistakes.

Now let's try a really simple C# program just to make sure everything is in order and the installation has worked. Enter the example shown in Figure 1-4 using your favorite editor, taking care to ensure you copy it exactly, as the C# language is case-sensitive. When done, save this in a folder called `/mono/Samples/Chapter1` under the name `helloworld.cs`. The `.cs` extension signals this is a C# file.

**Figure 1-4.** *The HelloWorld C# application*

Don't worry about trying to understand the syntax at this point; I'll provide a C# primer as part of the book. If you do understand the general notation, well done—you'll have a head start. Once you have saved the file, you can compile this into an executable and try executing it. So, first compile the file using the C# compiler. Enter the following line at a command-line prompt:

```
mcs helloworld.cs
```

As is often the case in the Linux world, no news is good news, and if you return to the command prompt with no further output, chances are it has worked. You can verify this by looking for the `helloworld.exe` file in the same folder. Now try running the executable by entering the following line:

```
mono helloworld.exe
```

The output should look similar to the following; in summary, you've written and implemented your first C# application—the infamous HelloWorld application:

```
[root@FedoraCore3 Chapter1]# mono helloworld.exe
Hello World!
```

How to Install Mono on Windows

One of the driving factors behind Mono is providing the comprehensive .NET environment on a platform other than Windows. So, if it is provided by Microsoft on the Windows platform, why bother to use Mono for Windows? A few good reasons exist, actually! First, if you want to test the interoperability of your Mono-developed applications on a multiplatform environment, it makes sense to test your applications under Mono on all the platforms you support—and Windows will most likely be one of them. Second, you may want to take advantage of the MonoDevelop environment for developing your applications without looking to purchase Microsoft Visual Studio. In addition, you may just love open source!

Whatever your reason, and for completeness, I'll provide installation instructions for the Windows platform. The first thing you'll notice is that Mono for Windows does not consist of lots of packages; it's a single executable that installs in a typical Windows way.

I'll show how to install the latest development release for Windows that is available, along with all other versions in the Downloads section of the Mono home page, at <http://www.mono-project.com/Downloads/>. The first step is to download the executable to a suitable folder; in my case it's under `\My Documents\My Mono\Packages`.

Note Mono for Windows is available only for Windows 2000 and above.

Once downloaded, you can start the process of installing Mono by simply executing the setup executable. This will start the setup process, and you should see an installation screen similar to the one shown in Figure 1-5.



Figure 1-5. *Mono for Windows setup screen*

At this point, you will be guided through the installation wizard, being prompted to answer questions along the way, such as the location of the program files. I suggest you accept the default values, as I will be assuming this throughout the book. You are free to provide your own answers to questions, but bear this in mind when you progress through the book, as the samples may assume the default values. I'll indicate that this is the case as you progress.

Once you've completed the wizard successfully, you should see a completion screen similar to Figure 1-6.



Figure 1-6. *Mono for Windows completion screen*

You can go through similar tasks as you did with the Linux version to see whether the installation was complete. The first step is to check that the command-line utilities have been installed; if you've accepted the default details as suggested, you should see a large number of files in the `C:\Program Files\Mono-1.1.4\bin` directory. These files are not only utilities but also runtime dynamic link libraries (DLLs) necessary to run Mono. You can check this by looking at the installation folder using Microsoft Windows Explorer; your screen should look something like Figure 1-7.

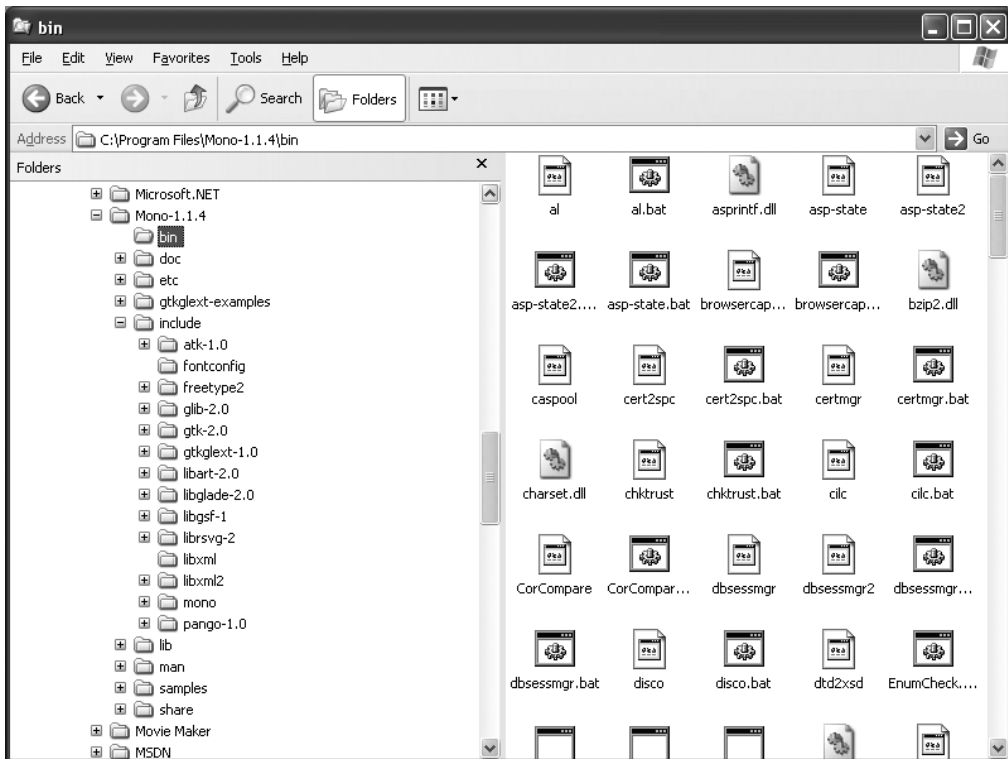


Figure 1-7. Windows Explorer view of Mono's binary directory

The second step is to test the Mono installation by writing, compiling, and executing your test application, `HelloWorld`. If these two tasks are successful, you can assume a successful installation and continue. So, create a source file using your favorite editor (I have used WordPad), and save this in the `My Documents\My Mono\Samples` directory as `helloworld.cs`, exactly as you would do (or did do) for the Linux test.

Once you've create this source file, you're ready to invoke the compiler. Start a command prompt window, and ensure that you're in the directory holding the `helloworld.cs` source file, as defined previously. From here you can issue the command to compile your source file, provided you have Mono's `\bin` directory within your `PATH` environment variable. If not, add this to your `PATH` environment variable before you start a command prompt window.

After changing to the directory and issuing the compile command, your window should look something like Figure 1-8. This demonstrates a successful compilation, and running the application will execute the application and display “Hello World!”

```

C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Mark>cd My Documents\My Mono\Samples
C:\Documents and Settings\Mark\My Documents\My Mono\Samples>cd
C:\Documents and Settings\Mark\My Documents\My Mono\Samples>
C:\Documents and Settings\Mark\My Documents\My Mono\Samples>mcs HelloWorld.cs
C:\Documents and Settings\Mark\My Documents\My Mono\Samples>dir
Volume in drive C has no label.
Volume Serial Number is 70B8-3015

Directory of C:\Documents and Settings\Mark\My Documents\My Mono\Samples
05/04/2005  21:47    <DIR>          -
05/04/2005  21:47    <DIR>          -
05/04/2005  21:45                169 HelloWorld.cs
05/04/2005  21:48                3,072 HelloWorld.exe
               2 File(s)                3,241 bytes
               2 Dir(s)      80,065,478,656 bytes free

C:\Documents and Settings\Mark\My Documents\My Mono\Samples>mono HelloWorld.exe
Hello World!

C:\Documents and Settings\Mark\My Documents\My Mono\Samples>

```

Figure 1-8. Successful compiler from the Windows command line

The Book's Project

Throughout the book, you will not only be introduced to what Mono consists of, and what the .NET Framework consists of, but you'll also be progressively taken through using your newfound knowledge to write a Really Simple Syndication (RSS) aggregator. RSS is a format for syndicating news headlines by using XML as its means of representing the information. During the book, you'll build a Windows Forms–based application that will allow you to subscribe to a list of RSS feeds and view the contents in an easy-to-read manner.

In addition to simply reading RSS feeds, the application will allow you to manage a list of your favorite feeds and will store the information offline in a MySQL database, allowing you to view the information when not connected to the Internet.

You can find the simple design for the application in the code download for this book. Refer to the Source Code section of the Apress Web site at <http://www.apress.com>.

Mono Support of .NET 2.0

So, where does that leave you with Mono? The Mono implementations defined within this book will work on versions 1.0, 1.1, and 2.0 of the .NET Framework with the latest Mono distribution, including support for version 2.0 elements of the .NET Framework and associated technologies. However, for clarity, most chapters will have a “What's New in Version 2.0?” section that will detail the key new features in the .NET Framework that will eventually (or may already be) supported by Mono.

Summary

In this chapter, I introduced the .NET Framework, its associated technologies, and the industry standards it has either embraced or defined. You should now have obtained and downloaded the version of Mono that is relevant to the platforms on which you want to run it and have the necessary knowledge to install and test Mono on your platform. You are now ready to begin Mono development!

In the next chapter, I'll introduce and discuss some of the tools that are either provided as part of the .NET Framework or provided as part of the open-source community, specifically an IDE, to make the process of developing applications easier and more graphical.