

Part I

C# Fundamentals

Chapter 1: The .NET Framework

Chapter 2: Getting Started with Visual Studio 2008

Chapter 3: C# Language Foundations

Chapter 4: Classes and Objects

Chapter 5: Interfaces

Chapter 6: Inheritance

Chapter 7: Delegates and Events

Chapter 8: Strings and Regular Expressions

Chapter 9: Generics

Chapter 10: Threading

Chapter 11: Files and Streams

Chapter 12: Exception Handling

Chapter 13: Arrays and Collections

Chapter 14: Language Integrated Query (LINQ)

Chapter 15: Assemblies and Versioning

The .NET Framework

The .NET Framework is a development framework created by Microsoft to enable developers to build applications that run on Microsoft (and other) platforms. Understanding the basics of the .NET Framework is essential because a large part of C# development revolves around using the classes in that framework.

This chapter explains the key components in the .NET Framework as well as the role played by each of the components. In addition, it examines the relationships among the various versions of the Framework, from version 1.0 to the latest 3.5.

What's the .NET Framework?

The .NET Framework has two components:

- Common Language Runtime
- .NET Framework class library

The Common Language Runtime (CLR) is the agent that manages your .NET applications at execution time. It provides core services such as memory, thread, and resource management. Applications that run on top of the CLR are known as *managed code*; all others are known as unmanaged code.

The .NET Framework class library is a comprehensive set of reusable classes that provides all the functionalities your application needs. This library enables you to develop applications ranging from desktop Windows applications to ASP.NET web applications, and Windows Mobile applications that run on Pocket PCs.

Common Language Runtime

The Common Language Runtime (CLR) is the virtual machine in the .NET Framework. It sits on top of the Windows operating system (Windows XP, Windows Vista, Windows Server 2008, and so on). A .NET application is compiled into a bytecode format known as MSIL.

Part I: C# Fundamentals

(Microsoft Intermediate Language). During execution, the CLR JIT (just-in-time) compiles the bytecode into the processor's native code and executes the application. Alternatively, MSIL code can be precompiled into native code so that JIT compiling is no longer needed; that speeds up the execution time of your application.

The CLR also provides the following services:

- ❑ Memory management/garbage collection
- ❑ Thread management
- ❑ Exception handling
- ❑ Security

.NET developers write applications using a .NET language such as C#, VB.NET, or C++. The MSIL bytecode allows .NET applications to be portable (at least theoretically) to other platforms because the application is compiled to native code only during runtime.

At the time of writing, Microsoft's implementation of the .NET Framework runs only on Windows operating systems. However, there is an open-source implementation of the .NET Framework, called "Mono," that runs on Mac and Linux.

Figure 1-1 shows the relationships between the CLR, unmanaged and managed code.

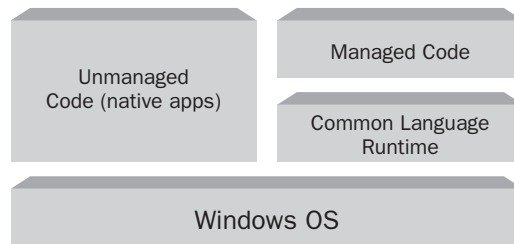


Figure 1-1

.NET Framework Class Library

The .NET Framework class library contains classes that allow you to develop the following types of applications:

- ❑ Console applications
- ❑ Windows applications
- ❑ Windows services

- ❑ ASP.NET Web applications
- ❑ Web Services
- ❑ Windows Communication Foundation (WCF) applications
- ❑ Windows Presentation Foundation (WPF) applications
- ❑ Windows Workflow Foundation (WF) applications

The library's classes are organized using a hierarchy of namespaces. For example, all the classes for performing I/O operations are located in the `System.IO` namespace, and classes that manipulate regular expressions are located in the `System.Text.RegularExpressions` namespace.

The .NET Framework class library is divided into two parts:

- ❑ * Framework Class Library (FCL)
- ❑ * Base Class Library (BCL)

The BCL is a subset of the entire class library and contains the set of classes that provide core functionalities for your applications. Some of the classes in the BCL are contained in the `mscorlib.dll`, `System.dll`, and `System.core.dll` assemblies. The BCL is available to all the languages using the .NET Framework. It encapsulates all the common functions such as file handling, database access, graphics manipulation, and XML document manipulation.

The FCL is the entire class library and it provides the classes for you to develop all the different types of applications listed previously.

Figure 1-2 shows the key components that make up the .NET Framework.

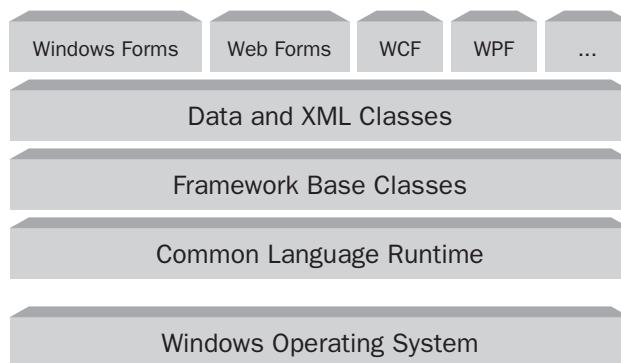


Figure 1-2

Assemblies and the Microsoft Intermediate Language (MSIL)

In .NET, an application compiled into MSIL bytecode is stored in an assembly. The assembly is contained in one or more PE (portable executable) files and may end with an EXE or DLL extension.

Some of the information contained in an assembly includes:

- ❑ **Manifest** — Information about the assembly, such as identification, name, version, and so on.
- ❑ **Versioning** — The version number of an assembly.
- ❑ **Metadata** — Information that describes the types and methods of the assembly.

Assemblies are discussed in more detail in Chapter 15.

To get a better idea of a MSIL file and its content, take a look at the following example, which has two console applications — one written in C# and the other written in VB.NET.

The following C# code displays the “Hello, World” string in the console window:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorldCS
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
            Console.ReadLine();
        }
    }
}
```

Likewise, the following VB.NET code displays the “Hello, World” string in the console window:

```
Module Module1

    Sub Main()
        Console.WriteLine("Hello, World!")
        Console.ReadLine()
    End Sub

End Module
```

When both programs are compiled, the assembly for each program has an `.exe` extension. To view the content of each assembly, you can use the `ildasm` (MSIL Disassembler) tool.

Launch the `ildasm` tool from the Visual Studio 2008 Command Prompt window (Start ⇨ Programs ⇨ Microsoft Visual Studio 2008 ⇨ Visual Studio Tools ⇨ Visual Studio 2008 Command Prompt).

The following command uses the `ildasm` tool to view the assemblies for the C# and VB.NET programs:

```
C:\MSIL>ildasm HelloWorldCS.exe
C:\MSIL>ildasm HelloWorldVB.exe
```

Figure 1-3 shows the content of the C# and VB.NET assemblies, respectively.

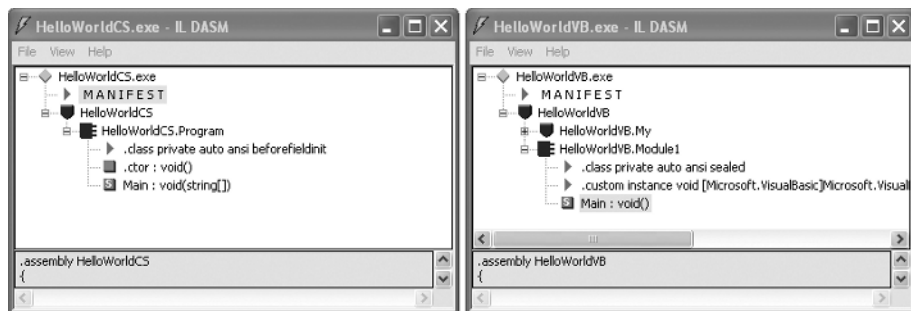


Figure 1-3

The `Main` method of the C# MSIL looks like this:

```
.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    // Code size      19 (0x13)
    .maxstack 8
    IL_0000: nop
    IL_0001: ldstr      "Hello, World!"
    IL_0006: call       void [mscorlib]System.Console::WriteLine(string)
    IL_000b: nop
    IL_000c: call       string [mscorlib]System.Console::ReadLine()
    IL_0011: pop
    IL_0012: ret
} // end of method Program::Main
```

Part I: C# Fundamentals

The `Main` method of the VB.NET MSIL looks very similar to that of the C# program:

```
.method public static void Main() cil managed
{
    .entrypoint
    .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
    // Code size      20 (0x14)
    .maxstack 8
    IL_0000: nop
    IL_0001: ldstr      "Hello, World!"
    IL_0006: call       void [mscorlib]System.Console::WriteLine(string)
    IL_000b: nop
    IL_000c: call       string [mscorlib]System.Console::ReadLine()
    IL_0011: pop
    IL_0012: nop
    IL_0013: ret
} // end of method Module1::Main
```

The important thing to note here is that regardless of the language you use to develop your .NET applications, all .NET applications are compiled to the MSIL bytecode as this example shows. This means that you can mix and match languages in a .NET project — you can write a component in C# and use VB.NET to derive from it.

Versions of the .NET Framework and Visual Studio

Microsoft officially released the .NET Framework in January 2002. Since then, the .NET Framework has gone through a few iterations, and at the time of writing it stands at version 3.5. While technically you can write .NET applications using a text editor and a compiler, it is always easier to write .NET applications using Visual Studio, the integrated development environment from Microsoft. With Visual Studio, you can use its built-in debugger and support for IntelliSense to effectively and efficiently build .NET applications. The latest version of Visual Studio is Visual Studio 2008.

The following table shows the various versions of the .NET Framework, their release dates, and the versions of Visual Studio that contain them.

Version	Version Number	Release Date	Versions of Visual Studio shipped
1.0	1.0.3705.0	2002-01-05	Visual Studio .NET 2002
1.1	1.1.4322.573	2003-04-01	Visual Studio .NET 2003
2.0	2.0.50727.42	2005-11-07	Visual Studio 2005
3.0	3.0.4506.30	2006-11-06	Shipped with Windows Vista
3.5	3.5.21022.8	2007-11-19	Visual Studio 2008

Starting with Visual Studio 2005, Microsoft dropped the .Net name from the Visual Studio.

The .NET Framework 3.5 builds upon version 2.0 and 3.0 of the .NET Framework, so it essentially contains the following components:

- ❑ .NET Framework 2.0 and .NET Framework 2.0 Service Pack 1
- ❑ .NET Framework 3.0 and .NET Framework 3.0 Service Pack 1
- ❑ New features in .NET 3.5

.NET Framework version 3.5 is dependent on .NET 2.0 and 3.0. If you have a computer with .NET 1.0, 1.1, and 2.0 installed, these three versions are completely separate from each other. When you install .NET 3.5 on a computer without the .NET Framework installed, it will first install .NET 2.0, followed by .NET 3.0, and then finally the new assemblies new in .NET 3.5.

Figure 1-4 summarizes the relationships between .NET 2.0, 3.0, and 3.5.

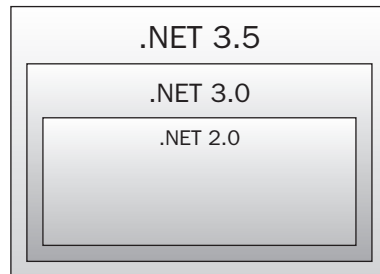


Figure 1-4

Summary

This chapter provided a quick overview of the .NET Framework and the various versions that make up the latest .NET Framework (3.5). Regardless of the language you use, all .NET applications will compile to a bytecode format known as MSIL. The MSIL is then JIT-compiled during runtime by the CLR to generate the native code to be executed by the processor.

In the next chapter, you start your journey to C# programming by learning use the development environment of Visual Studio 2008.

