ch01.fm Page 3 Wednesday, February 27, 2002 10:21 PM

FOR PUBLIC RELEASE

O N E

.NET Framework

Microsoft's popular programming language, Visual Basic, has been a favorite choice of programmers for many years. The ease with which Windows applications may be built, coupled with its wealth of database capabilities, has entrenched it in the hearts of many programmers. In its latest version, Microsoft has revamped Visual Basic, now called Visual Basic.NET (or simply VB.NET), to include full object-oriented capabilities and has provided support to work with the .NET Framework. We examine some of these issues throughout the book as we learn of the powerful services that are provided to the VB.NET programmer by the .NET Framework. In this chapter, we introduce the .NET Framework in sufficient detail so that you can immediately begin programming in VB.NET. For more in-depth information about .NET, you can refer to other books in The Integrated .NET Series from Object Innovations and Prentice Hall PTR. Of particular interest to VB.NET programmers will be Application Development Using Visual Basic and .NET *(Oberg, Thorsteinson, Wyatt), which delves into many important topics that are beyond the scope of this book.*

.NET: What You Need to Know

A beautiful thing about .NET is that, from a programmer's perspective, you scarcely need to know anything about it to start writing programs for the .NET environment. You write a program in a high-level language such as VB.NET, a

ch01.fm Page 4 Wednesday, February 27, 2002 10:21 PM

compiler creates an executable (.EXE) file, and you run that EXE file. We show you exactly how to do that in just a few pages. Naturally, as the scope of what you want to do expands, you will need to know more. But to get started, you need to know very little.

Even very simple programs, if they perform any input or output, will generally require the use of the services found in *library* code. A large library, called the .NET Framework Class Library, comes with .NET, and you can use all of the services of this library in your programs.

What is *really* happening in a .NET program is somewhat elaborate. The EXE file that is created does not contain executable code, but rather *Intermediate Language* code, or IL (sometimes called Microsoft Intermediate Language or MSIL). In the Windows environment, this IL code is packaged up in a standard portable executable (PE) file format, so you will see the familiar EXE extension (or, if you are building a component, the DLL extension). When you run the EXE, a special runtime environment (the Common Language Runtime or CLR) is launched, and the IL instructions are executed by the CLR. Unlike some runtimes, where the IL would be interpreted each time it is executed, the CLR comes with a just-in-time (JIT) compiler that translates the IL to native machine code the first time it is encountered. On subsequent calls, the code segment runs as native code.

Thus, in a nutshell, the process of programming in the .NET environment goes like this:

1. Write your program in a high-level .NET language such as VB.NET.
2. Compile your program into IL.
3. Run your IL program, which launches the CLR to execute your IL, using its JIT to translate your program into native code as it executes.

Installing the .NET SDK

All you need to compile and run the programs in this book is the .NET Framework SDK. This SDK is available on CD, or it can be downloaded for free from the Microsoft .NET Web site at *http://msdn.microsoft.com/net/*. Follow the installation directions for the SDK, and make sure that your computer meets the hardware requirements. (Generally, for the SDK, you need a fast Pentium processor and at least 128M of RAM.) Part of the installation is a Windows Component Update, which will update your system, if necessary, to recent versions of programs such as Internet Explorer. The SDK will install tools such as compilers, documentation, sample programs, and the CLR.

The starting place for the SDK documentation is the .NET Framework SDK Overview (see Figure 1–1).

ch01.fm Page 5 Wednesday, February 27, 2002 10:21 PM

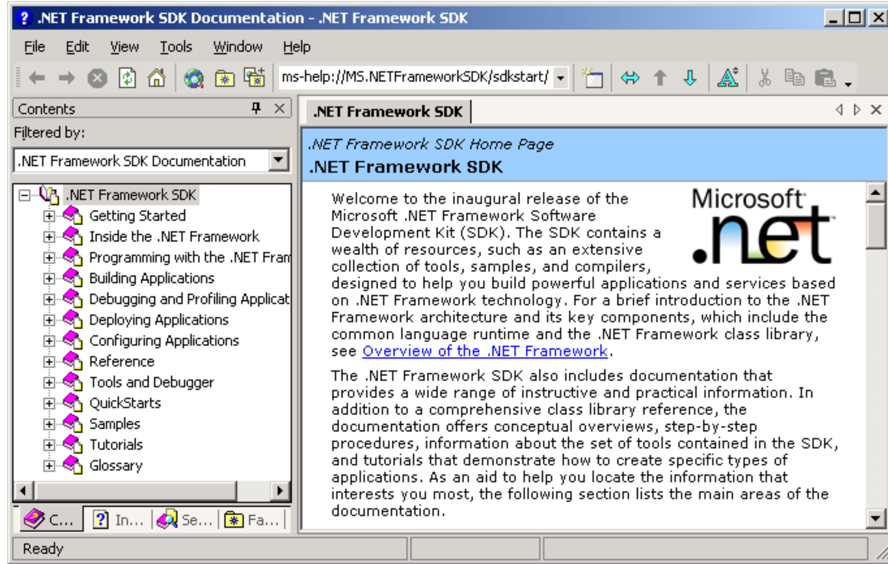
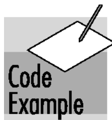


FIGURE 1–1 *Homepage of .NET Framework SDK.*

Installing the Book Software

The example programs found in this book are available on the Web at *http://www.objectinnovations.com/dotnet.htm/*. Download the file **Install_IntroVb.exe** and run this self-extracting file. If you accept the suggested installation directory, the software will be installed in the directory **OI\IntroVb** on your C: drive. There are subdirectories for each chapter of the book. The directory for Chapter 1 is **Chap01.** Sample programs are in named subdirectories of the chapter directory, and we will refer to these sample programs simply by name, such as **Hello.**

Your First VB.NET Program



Although we won't actually start to examine the structure of VB.NET programs until Chapter 2, you don't have to wait to compile and run your first VB.NET program. Start at the command prompt, and navigate to the **Hello** directory for this chapter. (If you accepted the default installation, the directory is **C:\OI\IntroVb\Chap01\Hello**.) The source file is **Hello.vb**. To compile this program, enter the following command:

```
>vbc hello.vb
```

ch01.fm Page 6 Wednesday, February 27, 2002 10:21 PM

The file **Hello.exe** will be created, which you can now run.

```
>hello
Hello World!
```

Setting Environment Variable

In order to run command line tools such as the VB.NET compiler using the name vbc rather than the complete path, certain environment variables must be set. The environment variables can be set using the batch file vsvars32.bat, which can be found in the Common\Tools directory of the Framework SDK.

If you have Visual Studio.NET installed, you can ensure that the environment variables are set up by starting your command prompt session from Start | Programs | Microsoft Visual Studio.NET 7.0 | Microsoft Visual Studio Tools | Microsoft Visual Studio.NET Command Prompt.

Visual Studio.NET

Although the .NET Framework SDK is all you need to compile and run VB.NET programs, the process will be much easier and more pleasant if you use the Visual Studio.NET integrated development environment (IDE). The IDE provides an easy-to-use editor, access to the compiler and debugger, and access to online help. We will discuss Visual Studio.NET in Chapter 3.

Understanding .NET

If you are eager to start learning the VB.NET programming language right away, by all means proceed directly to Chapter 2. The nice thing about a high-level programming language is that, for the most part, you do not need to be concerned with the platform on which the program executes (unless you are making use of services provided by the platform). You can work with the abstractions provided by the language and with functions provided by libraries.

However, you will better appreciate the VB.NET programming language and its potential for creating sophisticated applications if you have a general understanding of .NET. The rest of this chapter is concerned with helping you to achieve such an understanding. We address three broad topics:

- What Is Microsoft .NET?
- .NET Framework
- Common Language Runtime

ch01.fm Page 7 Wednesday, February 27, 2002 10:21 PM

What Is Microsoft .NET?

In this section, we answer the high-level question "What is .NET?" In brief, .NET represents Microsoft's vision of the future of applications in the Internet age. .NET provides enhanced interoperability features based upon open Internet standards.

The classic Windows desktop has been plagued by robustness issues. .NET represents a great improvement. For developers, .NET offers a new programming platform and superb tools.

XML plays a fundamental role in .NET. Enterprise servers (such as SQL 2000) expose .NET features through XML.

Microsoft .NET is a new platform at a higher level than the operating system. Three years in the making before public announcement, .NET is a major investment by Microsoft. .NET draws on many important ideas, including XML, the concepts underlying Java, and Microsoft's Component Object Model (COM). Microsoft .NET provides the following:

- A robust runtime platform, the CLR
- Multiple language development
- An extensible programming model, the .NET Framework, which provides a large class library of reusable code available from multiple languages
- A networking infrastructure built on top of Internet standards that supports a high level of communication among applications
- A new mechanism of application delivery, the Web service, that supports the concept of an application as a service
- Powerful development tools

Microsoft and the Web

The World Wide Web has been a big challenge to Microsoft. It did not embrace it early. But the Web actually coexists quite well with Microsoft's traditional strength, the PC. Using the PC's browser application, a user can gain access to a whole world of information. The Web relies on standards such as HTML, HTTP, and XML, which are essential for communication among diverse users on a variety of computer systems and devices.

The Windows PC and the Internet, although complex, are quite standardized. However, a Tower of Babel exists with respect to the applications that try to build on top of them: multiple languages, databases, and development environments. The rapid introduction of new technologies has created a gap in the knowledge of workers who must build systems using these technologies. This provides an opening for Microsoft, and some of the most talked about parts of .NET are indeed directed toward the Internet.

ch01.fm Page 8 Wednesday, February 27, 2002 10:21 PM

.NET provides many features to greatly enhance our ability to program Web applications, but this topic is beyond the scope of this book. For more information, please consult the following two books in The Integrated .NET Series:

- *Application Development Using Visual Basic and .NET* (Oberg, Thorsteinson, Wyatt)
- *Fundamentals of Web Applications Using .NET and XML* (Bell, Feng, Soong, Zhang, Zhu)

Windows on the Desktop

Microsoft began with the desktop, and the company has achieved much. The modern Windows environment has become ubiquitous. Countless applications are available, and most computer users are at least somewhat at home with Windows. There is quite a rich user interface experience, and applications can work together. But there are also significant problems.

PROBLEMS WITH WINDOWS

One of the most troublesome problems is the maintenance of applications on the Windows PC. Applications consist of many files, registry entries, shortcuts, and so on. Different applications can share certain DLLs. Installing a new application can overwrite a DLL that an existing application depends on, possibly breaking the older application (which is known as "DLL hell"). Removing an application is complex and often is imperfectly done. Over time, a PC can become less stable, and the cure eventually becomes reformatting the hard disk and starting from scratch.

There is tremendous economic benefit to using PCs, because standard applications are inexpensive and powerful, the hardware is cheap, and so on. But the savings are reduced by the cost of maintenance.

A ROBUST WINDOWS ENVIRONMENT

.NET has many features that will result in a much more robust Windows operating system. Applications no longer rely on storing extensive configuration data in the registry. In .NET, applications are self-describing, containing *metadata* within the program executable files themselves. Different versions of an application can be deployed *side-by-side*.

Applications run *managed code*. Managed code is not executed directly by the operating system, but rather by the special runtime—the CLR. The CLR can perform checks for type safety, such as for array out-of-bounds and memory overwrites. The CLR performs memory management, including automatic garbage collection, resulting in sharp reduction of memory leaks and similar problems.

ch01.fm Page 9 Wednesday, February 27, 2002 10:21 PM

Languages such as VB.NET and C# (pronounced "C sharp"), but not C++, can produce managed code that is verifiably secure. Managed code that is not verifiable can run if the security policy allows the code to ignore the verification process.

A New Programming Platform

.NET provides a new programming platform at a higher level than the operating system. This level of abstraction has many advantages:

- Code can be validated to prevent unauthorized actions
- It is much easier to program than the Win32 API or COM
- All or parts of the platform can be implemented on many different kinds of computers (as has been done with Java)
- All the languages use one class library
- Languages can interoperate with each other

We outline the features of this new platform, the *.NET Framework,* in the next section.

.NET Framework Overview

The .NET Framework consists of the CLR, the .NET Framework Class Library, the Common Language Specification (CLS), a number of .NET languages, and Visual Studio.NET. The overall architecture of the .NET Framework is depicted in Figure 1–2.

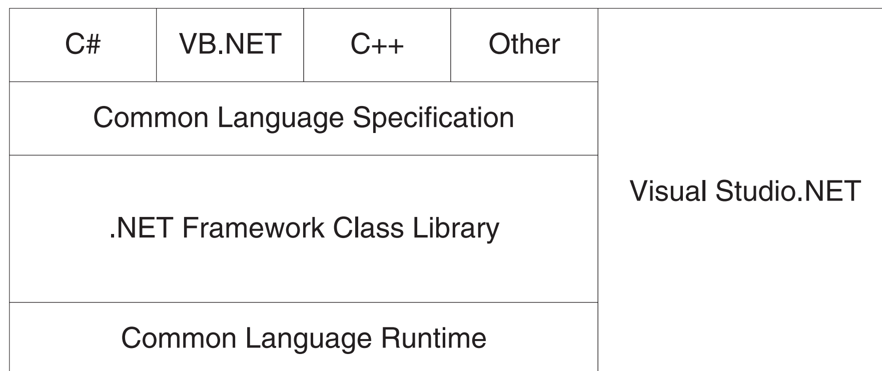


FIGURE 1–2 *Overall block diagram of .NET Framework.*

Common Language Runtime

A runtime provides services to executing programs. Traditionally, different programming environments have different runtimes. Examples of runtimes include the standard C library, MFC, the Visual Basic runtime, and the Java Virtual Machine (JVM).

The runtime environment provided by .NET, the CLR, manages the execution of code and provides useful services. The services of the CLR are exposed through programming languages. The syntax for these services varies from language to language, but the underlying execution engine providing the services is the same.

Not all languages expose all the features of the CLR. The language with the best mapping to the CLR is the new language C#. VB.NET, however, does an admirable job of exposing the functionality.

.NET Framework Class Library

The .NET Framework class library is huge, comprising more than 2,500 classes. All this functionality is available to all the .NET languages. The library (see Figure 1–3) consists of four main parts:

1. Base class library (which includes networking, security, diagnostics, I/O, and other types of operating system services)
2. Data and XML classes
3. Windows UI
4. Web services and Web UI

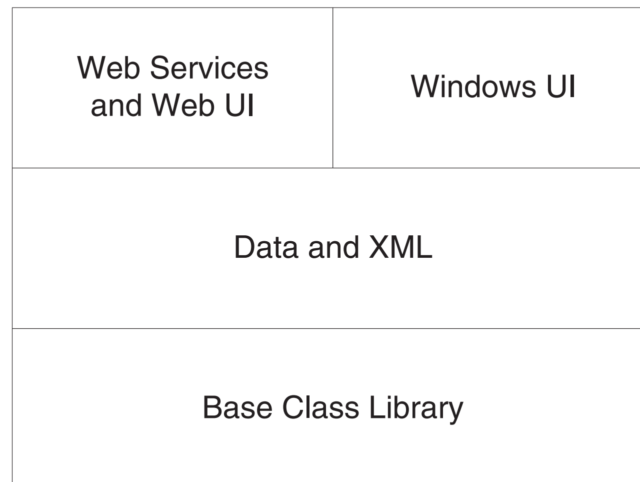


FIGURE 1–3 *Block diagram of .NET Framework Class Library.*

ch01.fm Page 11 Wednesday, February 27, 2002 10:21 PM

Common Language Specification

An important goal of the .NET Framework is to support multiple languages. But all languages are not created equal, so it is important to agree upon a common subset that all languages will support. The CLS is an agreement among language designers and class library designers about those features and usage conventions that can be relied upon.

CLS rules apply to public features that are visible outside the assembly where they are defined. (An assembly can be thought of as a logical EXE or DLL and will be discussed later in this chapter.) For example, the CLS requires that public names do not rely on case for uniqueness, because some languages are not case sensitive. For more information, see "Cross Language Interoperability" in "Inside the .NET Framework" in the .NET Framework SDK documentation.

Languages in .NET

A language is a CLS-compliant *consumer* if it can use any CLS-compliant type—that is, if it can call methods, create instances of types, and so on. (A type is basically a class in most object-oriented languages, providing an abstraction of data and behavior, grouped together.) A language is a CLS-compliant *extender* if it is a consumer and can also extend any CLS-compliant base class, implement any CLS-compliant interface, and so on.

Microsoft itself is providing four CLS-compliant languages. VB.NET, C#, and C++ with Managed Extensions are extenders. JScript.NET is a consumer.

Third parties are providing additional languages (more than a dozen so far). Active-State is implementing Perl and Python. Fujitsu is implementing COBOL. It should be noted that at present some of these languages are not .NET languages in the strict sense. For example, ActiveState provides a tool called PerlNET that will create a .NET component from a Perl class. This facility enables .NET applications to call the wealth of Perl modules, but it does not make Perl into either a consumer or an extender. For more information on PerlNET, see the book *Programming Perl in the .NET Environment* (Saltzman, Oberg), another book in The Integrated .NET Series.

Common Language Runtime

In this section, we delve more deeply into the structure of .NET by examining the CLR. We look at the design goals of the CLR and discuss the rationale for using managed code and a runtime. We outline the design of the CLR, including the concepts of MSIL, metadata, and JIT compilation. We compare the CLR with the Java Virtual Machine. We discuss the key concept in .NET of assem-

ch01.fm Page 12 Wednesday, February 27, 2002 10:21 PM

bly, which is a logical grouping of code. We explore the central role of types in .NET and look at the Common Type System (CTS). We explain the role of managed data and garbage collection. Finally, we use the Intermediate Language Disassembler (ILDASM) tool to gain some insight into the structure of assemblies.

Design Goals of the CLR

The CLR has the following design goals:

- Simplify application development
- Support multiple programming languages
- Provide a safe and reliable execution environment
- Simplify deployment and administration
- Provide good performance and scalability

SIMPLE APPLICATION DEVELOPMENT

With more than 2,500 classes, the .NET Framework class library provides enormous functionality that the programmer can reuse. The object-oriented and component features of .NET enable organizations to create their own reusable code. Unlike COM, the programmer does not have to implement any plumbing code to gain the advantages of components. Automatic garbage collection greatly simplifies memory management in applications. The CLR facilitates powerful tools such as Visual Studio.NET that can provide common functionality and the same UI for multiple languages.

MULTIPLE LANGUAGES

The CLR was designed from the ground up to support multiple languages. This feature is the most significant difference between .NET and Java, which share a great deal in philosophy.

The CTS makes interoperability between languages virtually seamless. The same built-in data types can be used in multiple languages. Classes defined in one language can be used in another language. A class in one language can even inherit from a class in another language. Exceptions can be thrown from one language to another.

Programmers do not have to learn a new language in order to use .NET. The same tools can work for all .NET languages. You can debug from one language into another.

SAFE EXECUTION ENVIRONMENT

With the CLR, a compiler generates MSIL instructions, not native code. It is this managed code that runs. Hence, the CLR can perform runtime validations on this code before it is translated into native code. Types are verified. Sub-

ch01.fm Page 13 Wednesday, February 27, 2002 10:21 PM

scripts are verified to be in range. Unsafe casts and uninitialized variables are prevented.

The CLR performs memory management. Managed code cannot access memory directly. No pointers are allowed. This means that your code cannot inadvertently write over memory that does not belong to it, possibly causing a crash or other bad behavior.

The CLR can enforce strong security. One of the challenges of the software world of third party components and downloadable code is that you open your system to damage from executing code from unknown sources. You might want to restrict Word macros from accessing anything other than the document that contains them. You want to stop potentially malicious Web scripts. You even want to shield your system from bugs of software from known vendors. To handle these situations, .NET security includes *Code Access Security* (CAS).

SIMPLER DEPLOYMENT AND ADMINISTRATION

With the CLR, the unit of deployment becomes an *assembly,* which is typically an EXE or a DLL. The assembly contains a *manifest,* which allows much more information to be stored.

An assembly is completely self-describing. No information needs to be stored in the registry. All the information is in one place, and the code cannot get out of sync with information stored elsewhere, such as in the registry, a type library, or a header file.

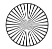
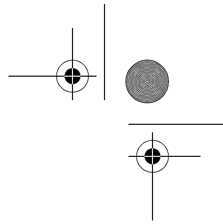
The assembly is the unit of versioning, so that multiple versions can be deployed side by side in different folders. These different versions can execute at the same time without interfering with each other.

Assemblies can be private or shared. For private assembly deployment, the assembly is copied to the same directory as the client program that references it. No registration is needed, and no fancy installation program is required. When the component is removed, no registry cleanup is needed, and no uninstall program is required. Just delete it from the hard drive.

In shared assembly deployment, an assembly is installed in the Global Assembly Cache (or GAC). The GAC contains shared assemblies that are globally accessible to all .NET applications on the machine. A download assembly cache is accessible to applications such as Internet Explorer that automatically download assemblies over the network.

PERFORMANCE

You may like the safety and ease-of-use features of managed code, but you may be concerned about performance. It is somewhat analogous to the concerns of early assembly language programmers when high-level languages came out.

The CLR is designed with high performance in mind. JIT compilation is designed into the CLR. The first time a method is encountered, the CLR performs verifications and then compiles the method into native code (which will contain safety features, such as array bounds checking). The next time the method is encountered, the native code executes directly.

Memory management is designed for high performance. Allocation is almost instantaneous, just taking the next available storage from the managed heap. Deallocation is done by the garbage collector, which Microsoft has tweaked for efficiency.

Why Use a CLR?

Why did Microsoft create a CLR for .NET? Let's look at how well the goals just discussed could have been achieved without a CLR, focusing on the two main goals of safety and performance. Basically, there are two philosophies. The first is compile-time checking and fast native code at runtime. The second is runtime checking.

Without a CLR, we must rely on the compiler to achieve safety. This places a high burden on the compiler. Typically, there are many compilers for a system, including third-party compilers. It is not robust to trust that every compiler from every vendor will adequately perform all safety checking. Not every language has features supporting adequate safety checking. Compilation speed is slow with complex compilation. Compilers cannot conveniently optimize code based on enhanced instructions available on some platforms but not on others. What's more, many features (such as security) cannot be detected until runtime.

Design of Common Language Runtime

So we want a runtime. How do we design it? One extreme is to use an interpreter and not a compiler at all. All the work is done at runtime. We have safety and fast builds, but runtime performance is very slow. Modern systems divide the load between the front-end compiler and the back-end runtime.

INTERMEDIATE LANGUAGE

The front-end compiler does all the checking it can do and generates an intermediate language. Examples include

- P-code for Pascal
- Bytecode for Java

The runtime does further verification based on the actual runtime characteristics, including security checking.

With JIT compilation, native code can be generated when needed and subsequently reused. Runtime performance becomes much better. The native

ch01.fm Page 15 Wednesday, February 27, 2002 10:21 PM

code generated by the runtime can be more efficient, because the runtime knows the precise characteristics of the target machine.

MICROSOFT INTERMEDIATE LANGUAGE

All managed code compilers for Microsoft .NET generate MSIL. MSIL is machine-independent and can be efficiently compiled into native code.

MSIL has a wide variety of instructions:

- Standard operations such as load, store, arithmetic and logic, branch, etc.
- Calling methods on objects
- Exceptions

Before executing on a CPU, MSIL must be translated by a JIT compiler. There is a JIT compiler for each machine architecture supported. The same MSIL will run on any supported machine.

METADATA

Besides generating MSIL, a managed code compiler emits metadata. Metadata contains very complete information about the code module, including the following:

- Version and locale information
- All the types
- Details about each type, including name, visibility, etc.
- Details about the members of each type, such as methods, the signatures of methods, etc.

Types

Types are at the heart of the programming model for the CLR. A type is analogous to a class in most object-oriented programming languages, providing an abstraction of data and behavior, grouped together. A type in the CLR contains the following:

- Fields (data members)
- Methods
- Properties
- Events

There are also built-in primitive types, such as integer and floating point numeric types, strings, etc. In the CLR, there are no functions outside of types, but all behavior is provided via methods or other members. We discuss types under the guise of classes and value types when we cover VB.NET.

Metadata is the “glue” that binds together the executing code, the CLR, and tools such as compilers, debuggers, and browsers. On Windows, MSIL

ch01.fm Page 16 Wednesday, February 27, 2002 10:21 PM

and metadata are packaged together in a standard Windows PE file. Metadata enables "Intellisense" in Visual Studio. In .NET, you can call from one language to another, and metadata enables types to be converted transparently. Metadata is ubiquitous in the .NET environment.

JIT COMPILATION

Before executing on the target machine, MSIL is translated by a JIT compiler to native code. Some code typically will never be executed during a program run. Hence, it may be more efficient to translate MSIL as needed during execution, storing the native code for reuse.

When a type is loaded, the loader attaches a stub to each method of the type. On the first call, the stub passes control to the JIT, which translates to native code and modifies the stub to save the address of the translated native code. On subsequent calls to the method, the native code is called directly.

As part of JIT compilation, code goes through a verification process. Type safety is verified, using both the MSIL and metadata. Security restrictions are checked.

COMMON TYPE SYSTEM

At the heart of the CLR is the Common Type System (CTS). The CTS provides a wide range of types and operations that are found in many programming languages. The CTS is shared by the CLR and by compilers and other tools.

The CTS provides a framework for cross-language integration and addresses a number of issues:

- Similar, but subtly different, types (for example, **Integer** is 16 bits in VB6, but **int** is 32 bits in C++; strings in VB6 are represented as BSTRs and in C++ as **char** pointers or a **string** class of some sort; and so on)
- Limited code reuse (for example, you can't define a new type in one language and import it into another language)
- Inconsistent object models

Not all CTS types are available in all languages. The CLS establishes rules that must be followed for cross-language integration, including which types *must* be supported by a CLS-compliant language. Built-in types can be accessed through the **System** class in the Base Class Library (BCL) and through reserved keywords in the .NET languages.

In Chapter 4, we begin our discussion of data types with the simple data types. We continue the discussion of types in Chapter 11, where we introduce *reference* types such as class and interface. At all times, you should bear in mind that there is a mapping between types in VB.NET, represented by keywords, and the types defined by the CTS, as implemented by the CLR.

ch01.fm Page 17 Wednesday, February 27, 2002 10:21 PM

Managed Data and Garbage Collection

Managed code is only part of the story of the CLR. A significant simplification of the programming model is provided through *managed data*. When an application domain is initialized, the CLR reserves a contiguous block of storage known as the *managed heap*. Allocation from the managed heap is extremely fast. The next available space is simply returned, in contrast to the C runtime, which must search its heap for space that is large enough.

Deallocation is not performed by the user program but by the CLR, using a process known as *garbage collection*. The CLR tracks the use of memory allocated on the managed heap. When memory is low, or in response to an explicit call from a program, the CLR "garbage collects" (or frees up all unreferenced memory) and compacts the space that is now free into a large contiguous block.

Summary

VB.NET does not exist in isolation, but has a close connection with the underlying .NET Framework. In this chapter, you received an orientation to the overall architecture and features of .NET.

Microsoft .NET is a new platform that sits on top of the operating system and provides many capabilities for building and deploying desktop and Web-based applications. .NET has many features that will create a much more robust Windows operating system.

The .NET Framework includes the Common Language Runtime (CLR), the .NET Framework Class Library, the Common Type System (CTS), the .NET languages, and Visual Studio.NET.

The CLR manages the execution of code and provides useful services. The design goals of the CLR included simple application development, safety, simple deployment, support of multiple languages, and good performance.

.NET uses managed code that runs in a safe environment under the CLR. .NET compilers translate source code into Microsoft Intermediate Language (MSIL), which is translated at runtime into native code by a just-in-time (JIT) compiler.

An assembly is a grouping of types and resources that work together as a logical unit. Types and the CTS are the heart of the CLR. Garbage collection is used by the CLR to automatically reclaim unreferenced data.

In Chapter 2, we will take our first steps in VB.NET programming.