CHAPTER

1

Introduction to the .NET Framework

Although the goal of this book is to introduce deployable solutions with the .NET Enterprise Servers, it is necessary to begin by examining the architecture of the .NET Framework. The .NET Enterprise Servers that will be introduced in later chapters will be discussed as tools that can be exploited by .NET developers. As a new, integrated software development environment, the .NET Framework allows the developer to combine the front-end development environment easily with the .NET Enterprise Servers to facilitate the development of scalable, Internet-ready applications. The framework is the foundation for the .NET programming environment and should be researched thoroughly by all developers who need to create new Windows applications.

The focus of this book is on building the .NET Framework by demonstrating the power that can be derived from the new .NET Enterprise Server architecture rather than on the technical aspects of the framework's components. For more information on the specifics of the .NET Framework, the reader can access Microsoft's Web site at www.microsoft.com/net.

This chapter first provides an overview of the .NET Framework. The framework is defined in a three-layer model. The model includes the common language runtime, the .NET Framework base classes, and the user and program interfaces. The section on user and program interfaces specifically defines the purpose of both Windows Forms and Web Forms.

After the overview of the .NET Framework, the chapter highlights some of the changes to the Visual Basic and Visual C++ programming languages. The chapter then introduces C# .NET and identifies its role in the new development framework. A short section discusses the changes to Active Server Pages (ASP). Finally, the chapter outlines the benefits of the .NET Framework to the developer.

By the end of this chapter the reader will have a deeper understanding of the role of the .NET Framework and the purpose for the various programming languages. The reader also will learn why the .NET Framework will become the solution of choice for Windows developers.

.NET Framework Overview

Microsoft's new software development platform, .NET Framework, is the first Microsoft development environment designed from the ground up for Internet development. Although .NET is not meant to be used exclusively for Internet development, its innovations were driven by the limitations of current Internet development tools and technology.

The basis of this new development platform consists of three primary components or layers: the common language runtime, the .NET Framework base classes, and the user and program interfaces, as demonstrated in Figure 1.1.

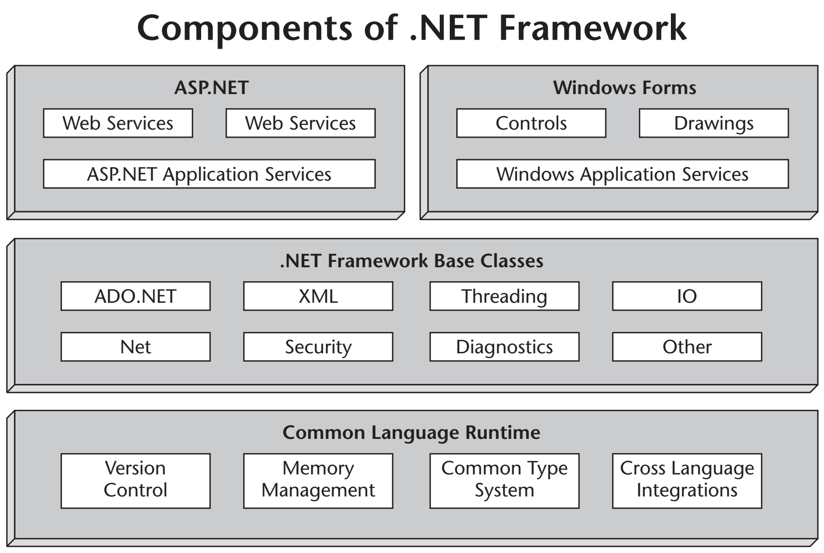

Figure 1.1 The .NET Framework has three layers: common language runtime, the .NET Framework base classes, and the user and program interfaces (ASP.NET and Windows Forms).

The foundation of the .NET Framework is the common language runtime. Its principal purpose is to load, execute, and manage code that has been compiled to Microsoft's new intermediate byte-code format called Intermediate Language (IL). Several languages, notably Microsoft's Visual Basic .NET and C# .NET (pronounced "C sharp"), have compilers supporting this format, and many more are currently in development. It is important to note that the IL code is not interpreted. The common language runtime uses just-in-time compilers to compile the IL code to native binary code before execution.

Other significant features of the common language runtime include the following:

- Version control
- Memory management
- Cross-language integration
- Common data type system

The next component or layer of the .NET Framework is the .NET Framework base classes. The purpose of this layer is to provide the services and object models for data access and manipulation, data streams (input/output [I/O]), security, thread management, and more. In many respects the Windows API (Application Programming Interface) has been abstracted into these base classes. These libraries are an object-oriented collection of classes that can be reused and enhanced to enable rapid application development. The classes support the creation of applications that range from ASP.NET Web pages and Web Services to traditional Windows and command line applications.

The .NET Framework also provides several runtime hosts, which are unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, creating a software environment that can exploit both managed and unmanaged features. The .NET Framework software developer's kit (SDK) not only provides several runtime hosts but also supports the development of third-party runtime hosts.

For example, ASP.NET hosts the runtime to provide a scalable, server-side environment for managed code. ASP.NET works directly with the runtime to enable .aspx pages and Web services, both of which are discussed later in this section.

The final layer of the .NET Framework consists of the user and program Interfaces. The two components of this layer are ASP.NET Application Services and Windows Application Services. The cornerstone of the ASP.NET Application Services is, of course, ASP.NET, which in turn supports the new Web services and Web Forms technologies that are discussed later. The Windows Application Services component supports traditional Windows programming applications through Windows Forms.

The following sections describe the main components of the .NET Framework in more detail.

Features of the Common Language Runtime

The common language runtime provides an execution environment that manages the set of code targeted to the .NET Framework. Code management can include memory management, thread management, security management, code verification, and compilation of the code. All managed components must first be assigned a level of trust. The level or degree of trust can vary on the basis of the following origins:

- Local computer
- Internet connection
- Enterprise local area network (LAN) connection

After the component is assigned an appropriate level of trust, the managed component either can or cannot perform functions from within the application. In fact, based on the trust levels, a managed component may act differently within the same active application. The degree of trust can be used to limit registry access, file access, or network functionality. The common language runtime enforces security in a way that enables a client to run executables without endangering or risking inappropriate access to sensitive local resources or devices. For example, a user could double-click an executable in an email, and although it may play a video or audio stream, access to personal data on the computer is not at risk. This enables an Internet application to be as feature-rich as normal desktop applications.

The common language runtime also uses a common type system (CTS) to enforce code robustness and language interoperability. Various Microsoft and other third-party compilers can generate managed code that complies with the CTS. The purpose of the CTS is to enforce a strict data type and code verification standard. Through the CTS, managed code can include managed classes, types, and other objects while enforcing type safety and conformity to the infrastructure.

Another primary goal of the common language runtime is increased programmer efficiency. With all the .NET Framework languages complying with the common language runtime, the programmer can choose a language of preference. Each language can take full advantage of the common language runtime, the class library, and the components. This will make future programming available in whichever language is best suited to the developer and application requirements while maintaining a high level of language interoperability and independence.

The built-in automatic memory management also enhances application and infrastructure stability. This new feature—the garbage collector—eliminates some of the most common application errors involving memory leaks and invalid memory references. It manages all references to objects and releases them when they are no longer in use. In a nutshell, the automatic memory management helps the application clean up after itself.

The common language runtime also should increase application performance. This may be accomplished in two ways: just-in-time (JIT) compliers and server-side applications. The JIT compilers are used to compile all managed code to native machine binary code at execution. Server-side applications can house application logic and business rules on .NET Enterprise Servers such as Internet Information Server (IIS) and SQL Server. This allows the consolidation of application logic and business rules in a more maintainable environment and permits execution closer to dependent resources. For many applications this will result in not only reduced maintenance requirements but increased responsiveness.

.NET Framework Class Library

The .NET Framework class library is a collection of reusable classes, or types, that tightly integrate with the common language runtime. .NET applications benefit from using and extending or inheriting the functionality from the classes and types available in the class library. The class library is very hierarchical and well organized, as shown in Figure 1.2. It starts with the most generic classes and continues to build down to classes with very specific and precise functionality. Although this library is extensive, its organization makes it easy to learn and use. In an age of ever-growing technology it is refreshing to see a new technology and a new architecture that promise a reduced learning curve. This model also makes it easy for third-party components to be integrated easily with the existing class library.

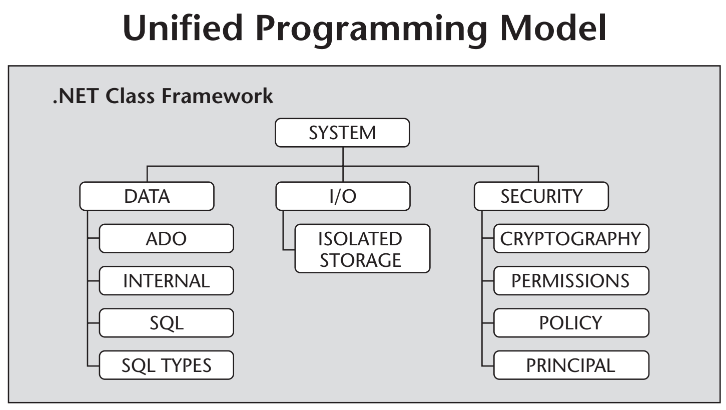

Figure 1.2 The .NET Framework class library is hierarchical in nature and goes from the more generic to the more specific.

As expected in an object-oriented class library, the .NET Framework classes enable developers to accomplish rapidly a wide range of common programming tasks, including things such as string management, file access, and database connectivity. Also, several classes facilitate highly specialized and custom development environments. These classes make the application development environment very flexible. The following types of applications are readily supported through the .NET Framework:

- ASP.NET applications
- Console applications
- Scripted applications
- Windows applications (Windows Forms)
- Web services

For example, the Windows Forms classes are used to create Windows graphical user interface (GUI) applications, which often are referred to as standalone applications. This is facilitated through a series of reusable graphical interface classes. Alternatively, in developing a Web-based application, the HTML and Web Forms classes facilitate its rapid development. Either way the underlying framework provides the flexibility for feature-rich applications regardless of the choice of application environment.

Client Application Development

Client applications represent the most common application environment in use today. These are the applications that one would invoke by opening some type of form and initiating an action. Examples of client applications range from word processing applications to a customized accounting package. One typically begins the application by opening a form from a desktop icon or the Windows Start menu. Client applications typically use the standard Windows features, such as dialogue boxes, menu items, and buttons. In most cases the files and executables for the application are stored and run locally and are interacted with by using local peripherals.

The traditional Windows programming environment has been replaced in .NET by the Windows Forms control. The managed Windows Forms control allows the application to be deployed over the Internet, and the user can view the application as a Web page.

In the past developers created such applications by using C or C++ in conjunction with the Microsoft Foundation Classes (MFC) or with a rapid application development (RAD) environment such as Microsoft Visual Basic. The .NET Framework incorporates aspects of the earlier products into a single, consistent development environment. The goal of the single environment is to simplify the development of client applications.

04235946 Ch01.F 12/11/02 11:43 AM Page 9

This simplicity can be further enhanced by incorporating the security and deployment features of the common language runtime. Because feature-rich applications can be deployed from the Web, many applications that once were installed on a user's system can now be Web-based. The application can define code-access security limits, which restrict what the application can do on an individual's machine. This gives the developer security features even though the application is running in the machine language. In fact, a single application can incorporate objects from multiple Web servers. Each object and Web server may have security rights different from those of another server. All security rights are evaluated even though the objects are used within a single application.

Windows Forms applications still have some advantages over Web-based applications. Windows Forms applications have a level of trust that is already assumed. This allows binary and natively executing code to interact with some of the resources on the local machine. This is used to perform the necessary GUI elements of the application.

Server Application Development

Web services are server-side application components that can be distributed easily in a similar fashion to Web sites. Web services components are not targeted for a specific browser and have no user interface. They are simply reusable software components designed to be used by other applications. Web-based applications, traditional applications, and even other Web services can use the Web services. The Web services are moving application development toward the Internet. All application development can now take advantage of the highly distributed world that previously was available only with Web-based applications.

ASP.NET is the environment that enables developers to use the .NET Framework to target Web services applications. Both Web Forms and Web services use IIS as the publishing mechanism for applications, and both have a collection of supporting classes in the .NET Framework.

If you are familiar with earlier versions of ASP technology, you will enjoy the new features of Web Forms. The most noticeable new feature is the ability to develop one's Web Forms in any language that supports the .NET Framework. You can use Visual Basic .NET to create your Web Forms if that is the language you prefer. You are not required to learn multiple languages to create different types of applications. Web Forms also can execute in the native machine language and take advantage of the common language runtime in the same manner as all other managed applications. This allows the Web Forms to be compiled and executed at a much faster rate. Users will notice that the ASP.NET pages are faster, more functional, and easier to create.

The .NET Framework also provides a collection of classes and tools to aid in the development and consumption of Web services applications. Web services are built on standards such as SOAP (a remote procedure-call protocol), XML (an extensible data format), and WSDL (Web Service Description Language). The .NET Framework conforms to these standards to promote interoperability with non-Microsoft solutions. Although many of the chapters in this book define the role of these protocols and show how they allow the .NET Framework to interact with .NET Enterprise Servers, readers can reference Chapter 15, “Web Services with .NET,” for more information on the role of each of these standards.

The .NET Framework provides a set of classes that conform to the underlying standards of SOAP, WSDL, and XML. Consequently, as the user develops and publishes Web services, he or she just focuses on the underlying logic and function of the service, not on infrastructure communication issues.

Web Forms and Web services are compiled and executed in native machine language and take advantage of the scalable architecture of IIS. While these technologies are dependent on IIS, their tight integration makes initial configuration and ongoing maintenance easy.

Windows Forms

As the technologies shift more toward the Web environment, it may appear that the .NET Framework is discouraging traditional Windows application development. This is far from accurate. The Windows Forms environment simplifies the development of applications and can be used to create feature-rich user interface Windows applications.

Windows Forms is a new forms package that can be used to develop and deploy Windows applications through the .NET Framework. Windows Forms is included as part of the new Microsoft Visual Studio package. It uses the .NET platform and leverages the new technologies, including a common application framework, managed execution environment, integrated security, and object-oriented design principles. In addition, Windows Forms offers full support for Web services and can connect easily to data stores by using a .NET data model called ADO.NET. The shared environment of Visual Studio allows a developer to create Windows Forms applications by using any of the languages that support the .NET platform. These languages include Visual Basic .NET, C++, and C#.

Windows Forms in the .NET Framework supports the development of Windows user interface applications. While the design is similar to that of Web Forms, the classes and implementation are significantly different. The development environment, however, is such that an application developer will feel equally comfortable writing an application using Windows Forms or Web Forms. Both environments have similar classes and some events in common, and the classes that are different are at least consistent in function and purpose. Although this book is not necessarily about creating Windows Forms or Web Forms, it is necessary to understand the foundation to be able to build on it with the .NET Enterprise Servers.

The primary goals of Windows Forms are ease of use and reusability. All Windows Forms are based on a common language runtime feature called delegates. Delegates are basically pointers. Each event can be given a delegate handler so that the user is not required to define the handler through another means (override, event map, and interface for all events on the class).

The process of creating a Windows Forms project is also easy. Creating a Windows Forms project with Visual Studio .NET creates only one project file that is compiled: Form1.cs. There are no header files, no interface definition files, no bootstrap application files, no resource files, and no library files. The goal is to keep the process minimal and easy to coordinate. All the information needed for the project is contained in the code for the form.

Because Windows Forms are built on the common language runtime, developers can choose any one of the many languages that are now targeting the common language runtime to build Windows applications. Developers can now write Windows Forms applications (or Web Forms applications or data applications) in a variety of languages that range from C# to COBOL, Eiffel, and Perl, among others. This decreases the learning curve by allowing developers to stick with their language of choice.

Windows Forms takes full advantage of GDI+, Microsoft's next-generation 2-D graphics system. The graphics programming model in Windows Forms is fully object-oriented, and the assorted pens, brushes, images, and other graphics objects follow the same ease-of-use guidelines as the rest of the .NET Framework. Developers can now include great new drawing features such as alpha blending, color gradients, textures, anti-aliasing, and vector image formats. When this feature is coupled with the Windows 2000 operating system's layered and transparent windows features, developers can create richer, more graphical Win32 applications with much less effort.

Web Forms

Web Forms are the forms that are engine-supplied with ASP.NET. Their purpose is to provide Web browser-based user interfaces. This technology has been enhanced to provide the next generation of development by adding features such as drag-and-drop development.

Web Forms consist of two parts: a template and a component. The template contains the layout information for the user interface elements. The component contains all the logic that is linked to the user interface. Web Forms take advantage of controls to make all the coordination required appear easy.

Controls run on the server and make themselves known to the client. This allows the application interface to have many of the characteristics of a normal Win32 application. Controls are reusable user interface elements that are used to construct the form.

New Programming Languages

Visual Studio .NET is Microsoft's .NET Framework development environment. Unlike earlier releases, this integrated development environment (IDE) is designed to support all programming languages that compile to Microsoft's Intermediate Language (MSIL), the "managed code" that actually is executed by the common language runtime. To meet this requirement, the Visual Studio .NET releases of Visual Basic and C++ underwent significant changes. In addition, Microsoft introduced C#, a new .NET programming language. To fully exploit the new capabilities of the .NET Framework, the changes to Visual Basic were particularly significant. While the .NET release is not compatible with prior versions of Visual Basic, Microsoft has incorporated a migration tool to ease the conversion of existing applications.

However, because the architecture of .NET is so different from the previous application development architecture, many applications may not port easily from the older platform to .NET. Consequently, many Visual Basic Version 6 applications will either be around for some time or undergo complete rewrites for the new .NET Framework.

The following sections introduce and reference the most popular new features of Visual Basic .NET, C#, and ASP.NET and discuss a couple of considerations for C++.

Visual Basic .NET

This release of Visual Basic represents the most significant architectural change since the initial release of Visual Basic. Because it now shares its development environment with other .NET languages, Visual Basic. NET can draw on the functionality of the .NET Framework base classes. Arguably the most significant advantage this gives Visual Basic developers is its tight integration with Web-based development. Visual Basic has long been the dominant programming language for Windows-based applications, but it always lacked an elegant approach to developing Internet applications. This release changes that situation. In addition to integration with the .NET Framework, Visual Basic .NET incorporates several new features, listed below, which expand the scalability and robustness of Visual Basic application development.

- Object-oriented development has been expanded to be fully supported in Visual Basic .NET. This includes full inheritance, parameterized constructors, and control over property and method overriding.
- Variables can be declared and initialized on the same line. Previously, variables were first declared (associated with a data type) and later initialized to a value. Additionally, the variant data type is no longer supported. However, the user still is not required to declare variables before initializing them. For example, the following ASP.NET Page directive can be used to turn off option explicit, thus suspending the need to declare variables before using them. Generally, this is a poor programming practice and is not encouraged.

  ```
  <%@Page Language = "vb" Explicit="false" %>
  ```
- Type Safety is used to generate errors when a data type conversion could fail at runtime.
- Error handling has been expanded to support the C-based language Try...Catch...Finally exception-handling statements. Visual Basic previously required, and still accepts, “On Error” statements in each procedure, which often results in duplicated code. The Try...Catch...Finally statements represent a more structured error-handling process that permits the nesting, and consequently the reuse, of exception-handling routines. Additionally, the Finally statement is a control structure for writing cleanup code that executes in both normal and exception conditions.
- Initializers can be used anywhere, including inside a control structure. Visual Basic .NET supports initializing the variable on the same line on which it is created. The semantics for a procedure-level declaration is similar if written in a single line as opposed to a declaration statement followed by an assignment statement. In other words, the following two examples are equivalent.

  ```
  Dim Y As Integer = 10

  Dim Y As Integer
  Y = 10
  ```
- Parentheses must be used in calling functions and subprocedures. All methods, functions, and subprocedures must now use parentheses to close the parameter list.
- Parameters are now passed to functions and subprocedures by value by default. Earlier versions of Visual Basic defaulted to passing parameters by reference. The ByRef keyword can still be used to pass a parameter by reference.
- New conversion methods are available for converting data types. Although older methods such as CStr are available, newer methods such as ToString are recommended.

These features combine with the .NET Framework to provide an easier environment to work with, a better performing application, and flexibility in incorporating Web-based applications. The resulting version of Visual Basic is more stable, scalable, and easier to use.

However, as previously noted, the downside to these much-anticipated new features is a lack of direct backward compatibility. Moderately complex applications cannot be ported easily from earlier versions to Visual Basic .NET. Although conversion tools exist, the number of required changes is significant. In many cases the migration tools simply flag an area that needs to be addressed, giving no indication of what the problem is or what needs to be done about it.

Visual C++

Among the programming languages Microsoft has altered to conform to the .NET Framework, Visual C++ has the fewest changes and requires the shortest learning curve. In addition, applications written in C++ can be ported easily to the new environment.

The most significant change to C++ revolves around the concept of "managed code," or the ability to compile code to MSIL for execution by the common language runtime. Within Visual Studio .NET, the developer can identify blocks of code as managed. The default option for Visual C++ development is still "unmanaged code," which is compiled directly to machine language code, lacks the type safety required for language interoperability, and forgoes the memory management features of the .NET Framework. In other words, Visual C++ developers can pretty much conduct business as usual. The language expands with a few additional items, which are detailed in the following list:

- Exception-handling tables
- Metadata describing the modules into which the code is compiled
- Location of object references (no need for CoCreateInstance; just use New to declare an object)

Visual C++ then uses managed code and lets the common language runtime handle the following items:

- Exception handling
- Security
- Lifetime management
- Debugging
- Profiling
- Memory management

C#

C# is Microsoft's programming language for its new .NET development environment. Microsoft's goal with C# is to provide a simple, modern, object-oriented .NET programming language that is Internet-centric. Although .NET code can be written in many languages, C# is the only language designed specifically for the .NET platform and for that reason may become the language of choice for this environment.

C# may be deceptively simple. Although it has only about 80 keywords and a dozen intrinsic data types, it is highly expressive. It includes support for all types of modern component-based, object-oriented development. C#, like C++ and Java, owes its origins to the C programming language. For that reason, C++ and Java developers will notice a striking similarity to those languages and enjoy an easy-to-learn and familiar programming environment.

Specifically, C# is an elegant language object-oriented language that:

- Readily supports the definition of and working with classes and their properties and methods.
- Implements encapsulation, inheritance, and polymorphism, the three pillars of object-oriented programming.
- Features self-contained class declarations (definitions). Header files or IDF (interface definition files) are not required to define or publicize a class's signature.
- Supports the implementation of interfaces, or contracts, to implement functionality. While a class can only inherit, or extend, from a single class, it can implement multiple interfaces.
- Supports structs, which are lightweight custom types that require fewer resources than does a more complex, robust type defined as a class.
- Provides for operator overloading and modern exception handling.

Microsoft again has "bet the company" on a new technology and developed C# as its premier language for that technology. C# was developed for the .NET platform and was modeled after the success of earlier languages. It has a rapid application development environment courtesy of Visual Basic, performance due to C, object-oriented nature from C++, and elegance from Java.

ASP.NET

ASP.NET is a programming framework developed by Microsoft for building powerful Web-based applications. While not a programming language, ASP.NET is the cornerstone of the .NET platform's Internet-centric orientation.

It is the latest version of Microsoft's ASP, a powerful, server-based technology for creating dynamic Web pages. Being a core element in the .NET platform, all .NET languages utilize ASP.NET as their entry point for Internet development.

Perhaps the biggest difference between earlier versions of ASP and ASP.NET is the way in which code runs on the Web server. With ASP.NET, code is compiled into executable classes that are compiled to native code by the common language runtime layer of the .NET Framework. All the earlier versions of ASP supported only scripting languages, which were interpreted. Since ASP.NET pages are now executable classes, they have access to all the other powerful features of the common language runtime, such as memory management, debugging, security, and strong data typing.

ASP.NET has built-in support for three languages: Visual Basic .NET, C#, and JScript.NET. The user can install support for other .NET-compatible languages as well. JScript.NET is Microsoft's latest version and implementation of JavaScript. Although this version still maintains its "scripting" feel, it is fully object-oriented and compiles to MSIL.

Because all ASP.NET code must be written in a .NET language, the ASP.NET programming framework is the foundation of Web services and Web Forms, the two components of ASP.NET Application Services. The following list illustrates how developers can exploit ASP.NET in developing Internet-centric applications.

- XML Web services gives developers the ability to access server functionality remotely. Using Web services, organizations can share their data and component libraries with other applications. Web services enable client/server and server/server communication through the HTTP protocol by using XML. Consequently, programs written in any language, using a component model and running on any operating system, can access Web services.
- ASP.NET Web Forms gives the developer the ability to create Web pages on the .NET platform. Web Forms enables developers to program embedded controls on either the client or the server. Using ASP Server Controls, Web applications enable the easy maintenance of state as information is communicated between the browser and the Web server.
- ASP.NET and the .NET Framework provide default authorization and authentication schemes for Web applications. Developers can easily remove, add to, or replace these schemes, depending on the needs of the application.
- Simple ASP pages can be migrated easily to ASP.NET applications. ASP.NET offers complete syntax and processing compatibility with ASP applications. Developers simply need to change the file extensions from .asp to. aspx to migrate their files to the ASP.NET page framework.

What .NET Means to Developers

Since the .NET Framework is a software development platform, it should be no surprise that .NET will have the biggest impact on developers. Here are some of the more significant benefits and implications awaiting them:

- With all the languages that soon will be compatible with .NET, developers have an unprecedented choice of interoperable languages to select from. That selection is far more than simply a lifestyle choice. .NET language choices provide the opportunity for a gradual transition from less complex languages to more powerful ones. Additionally, much of what is learned about the .NET base classes in one language carries over to the next language. Not only is the choice of languages unprecedented, so is the ease of the learning curve. Previously, Windows application programming required the developer to interact with components, application programming interfaces, and language-specific runtime libraries. This resulted in application development that was vastly different from one language to another. The new methodology, as described in Figure 1.3, allows all languages to interface directly with the .NET Framework.

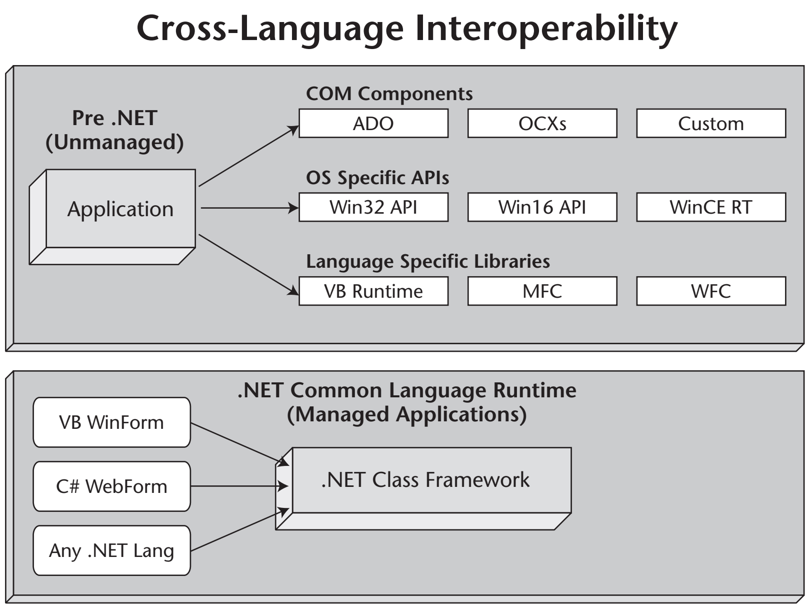

Figure 1.3 The .NET Framework overcomes previous weaknesses by allowing all languages to interface with the same common language runtime.

- The .NET languages are interoperable, meaning that classes developed in one .NET language can be freely reused and extended by a developer using another .NET language. While organizations may strive to standardize on a consistent development platform and language, developers may have more freedom to strike a proper balance between the skill required and the power necessary to accomplish their tasks.
- .NET continues the evolution of Windows development in the direction of providing more built-in functionality. The .NET base classes in particular encapsulate many of the mundane chores necessary in building application and network infrastructures. This frees the developer to focus more directly on solving business challenges.
- Developers may finally be freed from .DLL hell. Prior development environments stored information about components separately from those components. Consequently, installing and upgrading components were often a problem. .NET maintains these metadata inside the component itself, preventing the data from becoming out of sync with the component. In fact, different versions of the same component can coexist without compromising the integrity of the applications that are using them. Metadata about the component are maintained in the manifest and metadata elements of the components assembly, as shown in Figure 1.4.

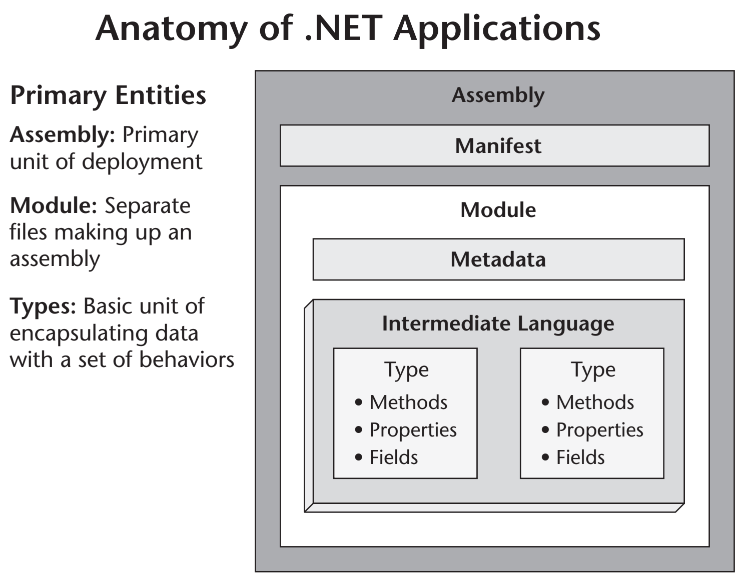
Anatomy of .NET Applications
Primary Entities
Assembly: Primary unit of deployment
Module: Separate files making up an assembly
Types: Basic unit of encapsulating data with a set of behaviors
Assembly
Manifest
Module
Metadata
Intermediate Language
Type
• Methods
• Properties
• Fields
Type
• Methods
• Properties
• Fields

Figure 1.4 The anatomy of a .NET application includes three primary entities: the assembly, the module, and the types.

- The common language runtime now features a garbage collector, a process responsible for removing objects from memory that no longer have references to them. The developer is freed from the tedious and often fruitless activity of trying to assure that all objects have been destroyed.
- Finally, .NET developers will have the challenge of working in a true object-oriented programming environment. While object-oriented programming and design can be a complex process, the elegant implementation offered to developers in the .NET environment will make it a thrilling and stimulating challenge.

Review Questions

1. What are the three layers of the .NET Framework?
2. What is the purpose of the common language runtime?
3. What is the role of the base class libraries?
4. How do Windows Forms differ from Web Forms?
5. What are the significant changes to ASP?
6. Why should I use the .NET languages?