CHAPTER 1

What Is .NET?

.NET is Microsoft's new initiative for building applications regardless of the platforms or languages in use. The .NET label applies to three distinct but related items: a vision for how information technology (IT) will evolve, a software platform to build .NET applications, and an application-hosting business designed to support the vision and market the platform. In this chapter, we inspect each of these items from a fairly nontechnical perspective. By the end of the chapter, the readers will have a good idea of where Microsoft is going with the .NET initiative and will understand the terminology, features, and services offered by the .NET Framework, the software platform for .NET applications.

INTRODUCTION

In June 2000, Microsoft announced the .NET initiative—a major shift in the technical direction for Microsoft and a major shift for those engaged in developing software based on Microsoft tools and technologies.

The label .NET has been applied to three distinct entities. They are:

1. A vision of how software will evolve to take advantage of the Internet and encompass the increasing variety of computing devices that are joining the PC in customers' offices, pockets, and homes.
2. A software platform to help developers build such applications and also to address some long-time shortcomings of Windows.
3. An application-hosting business that will deliver applications as services over the Internet.

In the rest of the chapter, we examine these three ideas in detail.

THE VISION

The Web has evolved a long way from browsing static Hypertext Markup Language (HTML) pages. Today, users can download music, participate in auctions, buy items online, and even talk to their family face-to-face over the Internet. Even businesses are not behind. They have been implementing business-to-business (B2B) and business-to-consumer (B2C) applications that communicate over the Internet.

Microsoft believes that the Internet will evolve from a collection of isolated Web sites and applications into a general "communication bus" for distributed applications. Individual parts of the distributed application could be running on different hardware and software platforms. The computing devices include your desktop systems as well as mobile devices such as cellular phones, Pocket PCs, personal digital assistants (PDAs), and so on. Even household appliances such as microwaves and dishwashers will participate in this communication over the Internet.

Web Services

To be fair, this vision of anytime, anywhere, any-device computing is also shared by many other software companies, such as IBM and Hewlett-Packard, and many respected computer scientists around the world. A key technology enabler for this distributed computing model is Web services. A *Web service* can be defined as a service that can be accessed programmatically over the Web. Companies can make their business applications available as Web services. These Web services, for example, can be used to integrate applications within various divisions of the same company. The Web services can also be used to automate communication over the Internet between two companies.

To be able to develop distributed applications that transcend geographical, hardware, and OS boundaries, Web services need to be based on universally accepted standards. Table 1.1 lays out the foundation elements of Web services.

TABLE 1.1. Web Services Foundation

Standard	*Purpose*
Internet	Ubiquitous communication
Extensible Markup Language (XML)	Universal data format
Simple Object Access Protocol (SOAP)	Communication protocol
Web Services Description Language (WSDL)	Describe the semantics of the methods available on a Web service
Universal Description, Discovery, and Integration (UDDI)	Publish and find Web services

In the "anywhere computing" vision, clients that wish to access Web services can be geographically distant from the servers. As the Internet has a broad geographical reach, it makes sense to deliver the services over the Internet.

To develop distributed client–server applications that transcend hardware and OS boundaries, Extensible Markup Language (XML) has been accepted as the universal language for defining data formats. XML provides a common data format that does not require business partners or customers to use a particular programming language, application, OS, or hardware.

XML by itself is not enough to achieve the client–server communication. To access a Web service, a client has to make a procedural call to the server, pass in the needed parameters, and get back the return value. A protocol has to be defined for such an exchange of information. To this effect, the W3C[1] has defined a protocol called Simple Object Access Protocol (SOAP). SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It specifies how a remote procedure call can be expressed in XML format. It is an XML-based protocol that consists of three parts:

1. An envelope that defines a framework for describing what is in a message and how to process it.
2. A set of encoding rules for expressing instances of application-defined data types.
3. A convention for representing remote procedure calls and responses.

Although the SOAP specification is independent of the underlying transport protocol, Hypertext Transport Protocol (HTTP) has been the sweet spot for the industry. Most companies let HTTP traffic pass through the firewall. Contrast this to other distributed object technologies such as Distributed Component Object Model (DCOM) and Common Object Request Broker Architecture (CORBA) that require opening ports on the firewall, thus compromising security.

Also note that although the client and the server can communicate with each other using raw SOAP packets, helper utilities are available on most platforms to hide the grunge work of creating SOAP packets:

1. The client makes a method call passing in the required parameters.

1. The World Wide Web Consortium (W3C) is a standards body that develops specifications to promote the evolution of the Web. More information on W3C can be found at *www.w3.org*.

2. A helper utility on the client side packages the method call and its parameters into a SOAP-compliant XML format and sends the SOAP packet to the remote server over a network protocol, preferably HTTP.
3. A helper utility on the server side unpackages the SOAP packet and calls the actual method, passing in the method parameters. On returning from the method, the utility repackages the return value into a SOAP packet and sends it back to the client.
4. The client-side utility unpackages the SOAP packet and returns the value to the client.

From a programming perspective, using the SOAP helper utilities makes calling a method to a remote system as simple as making a local method call.

Why is SOAP important? Because it provides the foundational invocation mechanism for application-to-application computing, irrespective of the underlying hardware or operating system platforms.

The SOAP specification is a work in progress. The current draft of the specification can be found at W3C's Web site [W3C-01].

Now we know how to make method calls on a Web service programmatically. However, we still don't know what methods are available as part of the Web service. We need a mechanism that describes the programmatic "interface" of the Web service; that is, the methods available on the Web service, the parameters for each method, and the return value of each method. A popular choice is to define this interface in Web Services Description Language (WSDL), an XML-based language that lets you express the functions and formats supported at any endpoint of the service. This programmatic interface is referred to as the *contract* of the Web service.

At this point, we know how to obtain method information on a Web service and how to make the method call. The remaining problem is to identify the server running the Web service.

It is likely that in some cases the server is known to the client. However, it is possible that the client is not particularly happy with the quality of the service or the cost of accessing the service, and may wish to use a different server. The beauty of the Web services programming model is that it doesn't matter which server provides the service, as long as the server adheres to the Web service contract. Coding-wise, all that is needed is to point to the right server. There is no change required to the rest of the code.

An industry-wide effort is underway to promote e-commerce among businesses. This project, called Universal Description, Discovery, and Inte-

gration (UDDI[2]), is an initiative to create an open framework for describing Web services, discovering businesses, and integrating business services over the Internet. UDDI enables business applications to do the following:

1. Discover each other.
2. Define how they interact over the Internet.
3. Share information in a global registry that will more rapidly accelerate the global adoption of B2B e-commerce.

Essentially, UDDI provides the "yellow pages" on the Internet for the industry. UDDI has also embraced SOAP and WSDL, making it convenient to obtain information from its repository programmatically.

Note that standards such as XML, SOAP, WSDL, UDDI, and so on, are not proprietary to Microsoft, although Microsoft has been a major contributor in driving these standards.

Microsoft's .NET initiative is built around XML, SOAP, and WSDL. The .NET technology and tools make it easy for companies to develop Web services and to consume other Web services.

Heterogeneous Environment

It is possible that Web services and other future applications may run on a variety of computing devices, not just PCs or mainframes. These devices need not run the same operating system. Microsoft Windows is not the only choice for the OS. Therefore, jointly with Intel and Hewlett-Packard, Microsoft has submitted the core .NET Framework specifications to European Computer Manufacturer's Association (ECMA[3]).This ECMA specification is referred to as the Common Language Infrastructure (CLI). The CLI specifications are not wedded to any OS. The .NET runtime is Microsoft's implementation of the CLI for Windows OS. However, Microsoft has also made available the source code to a working implementation of ECMA CLI that builds and runs on `FreeBSD`, a variation of the UNIX OS. Currently, there are various other initiatives underway to implement CLI on other variations of UNIX such as Linux.

Among other things, the CLI also specifies that a CLI-compliant application must run on different platforms without being rewritten for each specific

2. Complete information on UDDI can be found at *www.uddi.org*.

3. ECMA is an international standards organization. Their purpose is to standardize information and communication systems. More information on ECMA can be found at *www.ecma.ch*.

platform. A .NET application, for example, can run on many processors and platforms (currently, only x86 compatible CPUs are supported) as long as no OS-specific calls are made. So, if things go as expected by various implementers of CLI, you will be able to take a .NET executable that is built on one OS and run it on many other Windows and non-Windows OSs.

Smart Devices

In the not so distant future, Microsoft expects that PCs will be joined by many new kinds of smart devices such as data-enabled wireless phones, handheld computers, tablet PCs, home appliances, and so on. If an application has to run on all these devices, the application will have to automatically adapt its user interface to the capabilities of the device it runs on. This not only means adapting to each device's display and input capabilities, but also supporting new modes of communication such as spoken language and handwritten text.

To support software development for the smart devices, Microsoft has announced to release a subset of the .NET Framework called the .NET Compact Framework.

Compelling User Experience

Microsoft believes that, in this new distributed computing world, the experience should be very simple and compelling for the end users. To provide such experience, Microsoft intends to host a set of foundation services or building-block services. These building-block services will act as a central repository of data for users, allowing them to store e-mail, calendar information, contacts, and other important data, and present this data as needed to other Web sites.

THE PLATFORM

Figure 1.1 shows an overview of the .NET platform.

The central component of the .NET platform is the .NET Framework. This consists of a runtime environment called the common language runtime and a set of supporting libraries. The runtime environment controls the installation, loading, and execution of .NET applications. The libraries provide code for common programming tasks, thus increasing developer productivity. The libraries also provide a layer over many OS APIs, providing an isolation from OS dependencies.

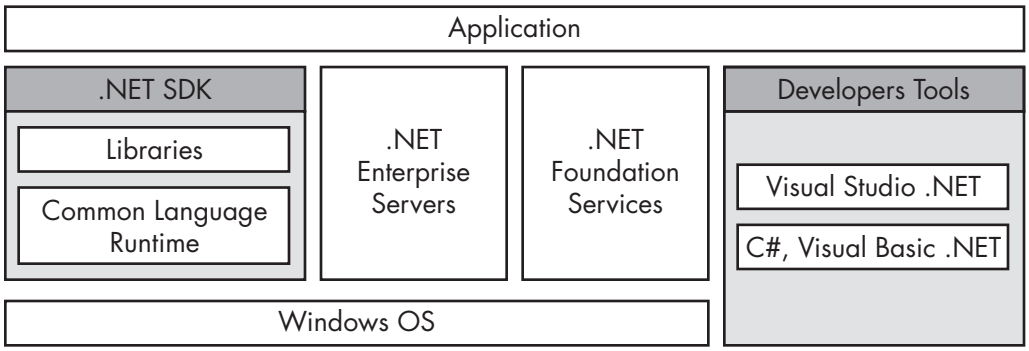

FIGURE 1.1 The .NET Platform.

Most enterprise applications and Web services require back-end servers to perform operations such as storing data, exchanging messages via e-mail, and so on. Microsoft's family of .NET servers such as SQL Server, Exchange Server, and so on, can be used to obtain such functionality. The family also includes some special servers that provide a higher level of integration and aggregation of Web services. BizTalk Server and Commerce Server, application frameworks for e-commerce, fall under this category.

The .NET platform also includes a set of developer tools such as Visual Studio .NET and programming languages such Visual Basic .NET and C# (pronounced *C sharp*).

In developing applications, developers can also take advantage of the foundation services offered by Microsoft or other software vendors. We take a look at a few important foundation services in a later section of this chapter.

Finally, the Windows operating system is at the base of the .NET platform. Operating systems such as Windows NT, Windows 2000, and Windows XP do not come preinstalled with the .NET Framework. However, one can install the framework separately by downloading it from Microsoft's Web site. Windows .NET and the newer releases of the Windows operating system are expected to ship with elements of the .NET vision.

THE .NET FRAMEWORK

The .NET Framework is a high-productivity, standards-based, multilanguage application execution environment. It consists of a runtime environment called the common language runtime and a set of libraries to boost programmers' productivity.

Common Language Runtime

The common language runtime, or just the runtime, is Microsoft's implementation of the ECMA CLI specification. When a .NET application is run, the common language runtime manages the loading and execution of the code and provides many runtime services to the application.

If you have been developing your code under COM, you will be surprised by the simplicity the .NET model offers. Forget dealing with GUIDs, CLSIDs, PROGIDs, IDL, type-libraries, apartments, server registration, `AddRef()`, `Release()`, and so on. They all have been replaced by a simpler model of programming.

It would not be fair to say that COM is dead. The basic tenet of COM, the ability for applications to communicate across hardware and programming language boundaries, is still present in .NET. In particular, the first release of the .NET Framework still depends on COM+ to provide enterprise services such as transaction and queuing. However, the COM infrastructure has certainly been replaced under .NET.

Besides providing a simpler model of communication, the .NET runtime provides many other services to simplify programming and to develop robust and secure applications. Any code that targets the .NET runtime is called the *managed code*—it can take advantage of the services offered by the .NET runtime. All other code, including existing legacy code, is called the *unmanaged code*. Although, the common language runtime can execute the unmanaged code, the code cannot avail the services provided by the common language runtime.

Let's examine some services provided by the common language runtime.

Simplified Deployment

In the simplest case, the directory hierarchy of an application can be copied to a machine and the application can be executed. There is no need to modify any registry entry. This is referred to as *XCOPY deployment*.

The framework also solves the "DLL hell" problem. A common problem with Windows is that upgrading a dynamic-link library (DLL) routinely breaks an already installed application. Under .NET, two versions of an application can execute side-by-side without breaking any application.

Hardware Independence

When a .NET application is built, the code is stored in a language called Microsoft Intermediate Language (MSIL). When the application is executed, the runtime performs a just-in-time (JIT) compilation to convert the MSIL code to machine instructions. This makes a .NET application run on any CPU

type, as long as a JIT compiler is available for the CPU. Moreover, the JIT compiler can perform hardware-specific optimizations, boosting execution performance.

Automatic Memory Management

When writing managed code, developers need not worry about memory deallocation issues. The runtime automatically frees any unused memory using a mechanism called *garbage collection*. Not only does this simplify programming, but it also makes the application more robust; as programmers sometimes simply forget to free previously allocated memory.

Cross-Language Integration

The .NET Framework defines a specification called the Common Language Specification (CLS). Among other things, the CLS defines a set of data types that is intended to work across all .NET-compliant programming languages. If these data types are used, the runtime provides seamless integration between applications developed in different programming languages. The integration is so seamless that a type defined in one language can be inherited in another language. Even exceptions can be thrown from one language and caught in another language.

Metadata Driven

An application developed for .NET contains complete information about the types it implements, the methods each type defines, the parameters for each method, and so on. The presence of such metadata eliminates the need for COM-style IDL and type libraries. This also makes it possible to keep the Windows registry clean.

Enhanced Security

.NET defines a permission-based security model that offers fine-grained control over which assembly can access what resource on the local machine. The security becomes especially important when users access code over the Internet. The runtime prevents the executions of any unauthorized code.

Interoperability

The runtime provides the functionality to integrate with legacy COM servers. The runtime also provides the ability to invoke any unmanaged code or Windows APIs (although such an application may not be portable to other platforms).

Class Libraries

The .NET Framework also provides hundreds of types (classes, interfaces, structures, etc.) that not only enable programmatic access to the features of the common language runtime, but also provide a number of useful high-level services to help developers boost their productivity. These types are collectively referred to as the .NET Framework Class Library.

The .NET Framework Class Library can roughly be broken down into four groups—the Base Class Library (BCL), ADO.NET and XML, Windows Forms, and ASP.NET.

The BCL implements the set of functionality that is shared by all the applications targeting the .NET Framework. It defines and implements all the core data types (e.g., string, integer, float, etc.) used by every application.

ADO.NET is the successor to an earlier data access technology called Active Data Object (ADO). ADO.NET provides a set of classes to access and manipulate data. The data is typically obtained from a database and can be converted to XML for better remote manipulation.

Windows Forms (often called WinForms) provide features for developing standard Windows desktop applications. They provide a rich, unified set of controls and drawing functions for all languages, effectively wrapping Windows user interface (UI) APIs in such a way that developers rarely would need to access the Windows APIs directly.

ASP.NET is the successor to a Web-request processing technology called Active Server Pages (ASP). ASP.NET adds two significant enhancements to ASP:

1. It simplifies the process of developing Web services.
2. It provides a model of developing a Web browser-based UI called Web Forms. Controls on the Web Forms run on the server but the UI is displayed on the client browser. This takes lots of coordination and behind-the-scenes activity. However, the end result is Web interfaces that look and behave very much like WinForms interfaces. Moreover, the Web interfaces can deal with a broad range of browsers such as Microsoft Internet Explorer as well as less capable browsers such as the ones found on wireless palmtop devices. WebForms will render themselves appropriately on the target device.

DEVELOPMENT TOOLS

A productive set of tools is critical to developer success on a new platform like .NET. Microsoft offers many development tools to build Web services as well as traditional Windows applications.

Programming Languages

.NET offers an improved ball game for programmers. Features such as automatic memory management make it unnecessary for programmers to deal with these issues. The .NET programming model encourages object-oriented programming.

To simplify programming under .NET and to exploit the capability of the .NET Framework to its fullest extent, Microsoft has introduced a new programming language called C#, which offers the simplicity of Visual Basic and the flexibility of C++. C# borrows most of its constructs directly from C++, making C++ and Java programmers feel right at home. More information about the origins of C# can be found in an interview with Anders Hejlsberg [Osb-00], the chief architect for C#.

Microsoft has also introduced Visual Basic .NET, an upgrade to its popular Visual Basic programming language. Visual Basic .NET adds object-oriented features as well as provides full access to .NET platform features. The new feature list of Visual Basic.NET can be found in [Pat-01a] and [Pat-01b].

Microsoft has also extended C++ to develop code for .NET. This extension is referred to as Managed Extension for C++.

Finally, .NET provides an open standard for developing language compilers that target .NET. Many independent software vendors are providing their own programming language support for .NET.

The .NET Framework SDK

The .NET Framework SDK contains documentation, tools, C# and Visual Basic .NET compilers, and samples for developers to write, build, test, and deploy .NET applications. The SDK also includes the .NET Framework as a redistributable package.

The SDK can be downloaded free of charge from Microsoft's Web site. Remember to read the licensing agreement when you download the SDK.

Visual Studio .NET

Visual Studio .NET is the next generation of Microsoft's popular multilanguage development tool, built especially for .NET. Visual Studio .NET helps you build as well as consume Web services and .NET applications quickly. It supports C#, Visual Basic .NET, and C++. Any other .NET programming language can be easily integrated into Visual Studio .NET. The Integrated Development Editor (IDE) contains many features, such as IntelliSense, to boost programmers' productivity. Visual Studio .NET will likely remain the most popular choice for developing .NET applications.

FOUNDATION SERVICES

Microsoft also envisions that providing a compelling user experience to consumers is important for the success of Internet as a communication bus. To this effect, Microsoft plans to release some foundation or building block Web services. Software vendors can leverage against these foundation services. With time, Microsoft intends to release more such foundation services.

The first set of foundation services are being released as a product called Microsoft .NET My Services. Table 1.2 lists these services.

TABLE 1.2. Microsoft .NET My Services

Name	*Description*
.NET Address	Billing, shipping, and other addresses
.NET Profile	Name, picture, and so on
.NET Contacts	Electronic address book
.NET Location	Electronic and geographical locations
.NET Alerts	Send and/or electronic notifications
.NET Inbox	E-mail and voice-mail storage
.NET Calendar	Appointment management
.NET Documents	Users can store, share, and back up important files
.NET ApplicationSettings	Application settings
.NET FavoriteWebSites	List of favorite Web sites
.NET Wallet	Credit card information, coupons, receipts, and so on
.NET Devices	Settings for various personal devices
.NET Services	List of services provided
.NET Usage	Usage report for the preceding services

A common theme behind Microsoft's foundation services is that the user information is stored at a central place and can be retrieved anytime, anywhere. These foundation services open the door for developing innovative software applications. For example, using .NET Calendar, a scheduling application at your doctor's Web site might be able to access your Web-hosted calendar to see when you are available, schedule an appointment at an appropriate time, and remind you using .NET Alerts (on your PC, pager, or any other notification device) when the appointment is approaching. As a Web service is based on open standards, the scheduling application can be developed for Windows, UNIX, or any other OS.

The foundation services are built around user identity. Microsoft provides a user authentication service called Microsoft Passport that deserves special attention.

User Authentication Service

Consumers do their Internet shopping on many Web sites. A common problem that they face today is that they are asked to enter account information, such as user name and password, on each Web site they visit because each Web site maintains its own database of customers.

Microsoft Passport promises a solution to this dilemma. Rather than signing up for an account on every Web site, the user signs up for a Passport account, either through *www.passport.com* or through related services like Hotmail (every Hotmail user automatically has a Passport account). The user can choose how much information to store in the MS Passport account—from a simple user name and password to a complete wallet with credit card information, shipping and billing addresses, and more.

Using Microsoft's single sign-in (SSI) service, a Passport member can use one sign-in name and password at all participating Web sites. Passport sign-in names are tied to individuals and not computers, members can access Passport sites from a wide range of devices.

WHAT DOES IT ALL MEAN?

Microsoft's .NET initiative impacts consumers, businesses, software vendors, and developers.

Consumers will be the biggest beneficiaries of .NET and the foundation services. As the data is stored on the Web, they will be able to access documents and other personal information anytime, anywhere, from any smart device.

For businesses, implementing applications using Web services solves many of today's B2B and B2C integration challenges.

By making their business applications available as Web services, or by providing innovative Web services, software vendors may be able to find newer modes of revenue. Microsoft itself is gravitating toward providing a subscription-based model for its services, thereby ensuring a monthly source of revenue.

The .NET Framework helps developers write robust, secure, Internet-enabled code in a language of their choice. The rich set of class libraries provided by the .NET Framework, as well as new features in Visual Studio .NET will boost developer productivity.

REFERENCES

[W3C-01] Gudgin, Martin, et al., "SOAP Version 1.2," W3C Working Draft, July 2001.
www.w3.org/TR/2001/WD-soap12-20010709/

[Osb-00] Osborn, John, "Deep Inside C#: An Interview with Microsoft Chief Architect Anders Hejlsberg," July 2000.
windows.oreilly.com/news/hejlsberg_0800.html

[Pat-01a] Pattison, Ted, "Basic Instincts: New Features in Visual Basic .NET," *MSDN Magazine*, May 2001.
msdn.microsoft.com/msdnmag/issues/01/05/instincts/instincts0105.asp

[Pat-01b] Pattison, Ted, "Basic Instincts: Exploiting New Language Features in Visual Basic .NET, Part 2," *MSDN Magazine*, August 2001.
msdn.microsoft.com/msdnmag/issues/01/08/Instincts/Instincts0108.asp