Part One

ASP.NET Introduction

2786c01.fm Page 2 Saturday, December 20, 2003 6:30 PM

2786c01.fm Page 3 Saturday, December 20, 2003 6:30 PM

CHAPTER 1

The .NET Framework

ONCE AGAIN, Microsoft is doing what it does best: creating innovative new technologies and wrapping them in marketing terms that cause widespread confusion. Just as developers are finally sorting out buzzwords like ActiveX and Windows DNA (Distributed interNet Architecture), Microsoft has created a whole new collection of jargon revolving around .NET. So exactly what does it all mean?

This chapter examines the technologies that underlie .NET. First, you'll take a quick look at the history of web development, and learn why the .NET Framework was created. Next, you'll get a high-level overview of the different parts of .NET, and see how ASP.NET fits into the wider world of development—and Microsoft's long-term plans.

The Evolution of Web Development

The Internet began in the late 1960s as an experiment. Its goal was to create a truly resilient information network—one that could withstand the loss of several computers without preventing the others from communicating. Driven by potential disaster scenarios (such as nuclear attack), the U.S. Department of Defense provided the initial funding.

The early Internet was mostly limited to educational institutions and defense contractors. It flourished as a tool for academic collaboration, allowing researchers across the globe to share information. In the early 1990s, modems were created that could work over existing phone lines, and the Internet began to open up to commercial users. In 1993, the first HTML browser was created, and the Internet revolution began.

HTML and HTML Forms

It would be difficult to describe early websites as web applications. Instead, the first generation of websites often looked more like brochures, consisting mostly of fixed HTML pages that needed to be updated by hand.

A basic HTML page is a little like a word-processing document—it contains formatted content that can be displayed on your computer, but it doesn't actually *do* anything. The following example shows HTML at its simplest, with a document that contains a heading and single line of text:

```
<html>
    <body>
        <h1>Sample Web Page Heading</h1>
        <p>This is a sample web page.</p>
    </body>
</html>
```

2786c01.fm Page 4 Saturday, December 20, 2003 6:30 PM

There are two types of content in an HTML document: your text, and the tags that tell the browser how to format it. The tags are easily recognizable, because they occur inside angled brackets (< >). The HTML language defines tags for different levels of headings, paragraphs, hyperlinks, italic and bold formatting, horizontal lines, and so on. For example, <h1>Some Text</h1> tells the browser to display "Some Text" in the Heading 1 style, which uses a large boldface font. Figure 1-1 shows the simple HTML page in a browser.

Figure 1-1. Static HTML: the "brochure" site

TIP You don't need to master HTML to program ASP.NET web pages, although it's often useful. For a quick introduction to HTML, refer to one of the excellent HTML tutorials on the Internet, such as `http://www.w3schools.com/html` or `http://archive.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html`.

HTML 2.0 introduced the first seed of web programming with a technology called *HTML forms*. HTML forms expand the HTML language so that it includes not only formatting tags, but also tags for graphical widgets, or *controls*. These controls include common ingredients like drop-down lists, text boxes, and buttons. Here's a sample web page created with HTML form controls:

2786c01.fm Page 5 Saturday, December 20, 2003 6:30 PM


```
<html>
    <body>
        <form>
            <input type="checkbox">This is choice #1<br>
            <input type="checkbox">This is choice #2<br><br>
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

In an HTML form, all the controls are placed between the <form> and </form> tags. The preceding example includes two check boxes (represented by the <input type="checkbox"> tag) and a button (represented by the <input type="submit"> tag). In a browser, this takes the appearance shown in Figure 1-2.

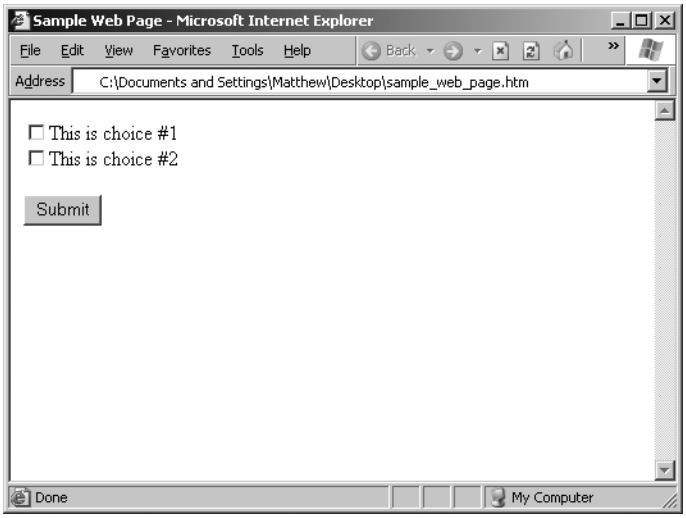


Figure 1-2. An HTML form

HTML forms allow web application developers to design standard input pages. When the user clicks the Submit button in the page shown in the previous example, all the data in the input controls (in this case, the two check boxes) is patched together into one long string and sent to the web server. On the server side, a custom application receives and processes the data. Amazingly enough, the controls that were created for HTML forms ten years ago are still the basic foundation that you'll use to build dynamic ASP.NET pages! The difference is the type of application that runs on the server side. In the past, when the user clicked a button on a form page, the information might have been emailed to a set account, or sent to an application on the server that used the challenging *CGI* (Common Gateway Interface) standard. Today, you'll work with the much more capable and elegant ASP.NET platform.

2786c01.fm Page 6 Saturday, December 20, 2003 6:30 PM

Server-Side Programming

In order to understand why ASP.NET was created, it helps to understand the problems of other web development technologies. CGI applications, for example, don't scale very well, which means they are hard pressed to support a large number of users. With a CGI application, the web server must launch a completely separate instance of the application for each web request. If the website is popular, the web server must struggle under the weight of hundreds of separate copies of the application, eventually becoming a victim of its own success. CGI applications also fare poorly when creating complex, integrated websites. Instead, CGI is best suited for "quick-and-dirty" website programming—in other words, when adding a couple of frills to a mostly static website.

To counter these problems, Microsoft developed *ISAPI* (Internet Server Application Programming Interface), a higher-level programming model. ISAPI solved the performance problem, but at the cost of significant complexity. Even after ISAPI developers master the tricky C++ programming language, they still lie awake at night worrying about confounding issues like multithreading. ISAPI programming is definitely not for the faint of heart.

ISAPI never really went away. Instead, Microsoft used it to build higher-level development platforms, like ASP and ASP.NET. Both of these technologies allow developers to program dynamic web pages without worrying about the low-level implementation details. For that reason, both platforms have become incredibly successful. The original ASP platform garnered a huge audience of nearly one million developers. When ASP.NET was released, it generated even more interest as the centerpiece of the .NET Framework. In fact, ASP.NET was enthusiastically put to work in dozens of large-scale commercial websites even when it was only in late beta.

Despite having similar underpinnings, ASP and ASP.NET are radically different. ASP is a script-based programming language that requires a thorough understanding of HTML, and a good deal of painful coding. ASP.NET, on the other hand, is an object-oriented programming model that lets you put together a web page as easily as you would build a Windows application. In many respects, it's easier to learn ASP.NET than to master ASP, even though ASP.NET is far more powerful.

Client-Side Programming

At the same time that server-side web development was moving through an alphabet soup of technologies, a new type of programming was gaining popularity. Developers began to experiment with the different ways they could enhance web pages by embedding multimedia and miniature applets built with JavaScript, DHTML (Dynamic HTML), and Java code. These client-side technologies don't involve any server processing. Instead, the complete application is downloaded to the client browser, which executes it locally.

2786c01.fm Page 7 Saturday, December 20, 2003 6:30 PM

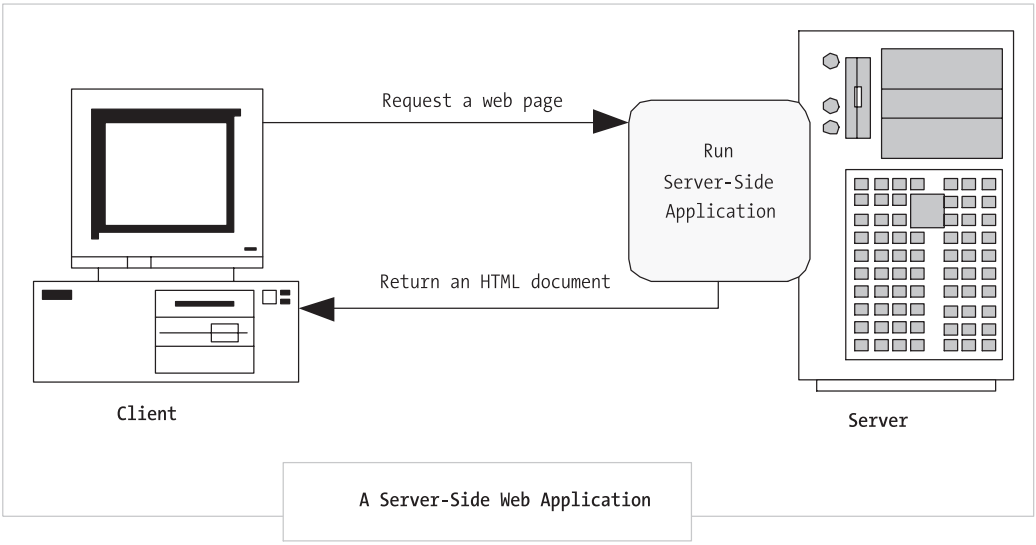
The greatest problem with client-side technologies is that they aren't supported equally by all browsers and operating systems. One of the reasons that web development is so popular in the first place is because web applications don't require setup CDs, downloads, and other tedious (and error-prone) deployment steps. Instead, a web application can be used on any computer that has Internet access. But when developers use client-side technologies, they encounter a few familiar headaches. Suddenly, cross-browser compatibility becomes a problem. Developers are forced to test their websites with different operating systems and browsers, and they might even need to distribute browser updates to their clients. In other words, the client-side model sacrifices some of the most important benefits of web development.

For that reason, ASP.NET is designed as a server-side technology. All ASP.NET code executes on the server. When the code is finished executing, the user receives an ordinary HTML page, which can be viewed in any browser. Figure 1-3 shows the difference between the server-side and client-side model.

There are other reasons for avoiding client-side programming. These include the following:

- **Isolation.** Client-side code can't access server-side resources. For example, there's no easy way for a client-side application to read a file or interact with a database on the server.
- **Security.** Client-side code can be viewed by the end user. And once users understand how an application works, they can tamper with it.
- **Thin clients.** As the Internet continues to evolve, new web-enabled devices like mobile phones, palmtop computers, and PDAs are appearing. These devices can communicate with web servers, but they don't support all the features of a traditional browser. Thin clients can use server-based web applications, but they won't support client-side features like JavaScript.

In some cases, ASP.NET allows you to combine the best of client-side programming with server-side programming. For example, the best ASP.NET controls can intelligently detect the features of the client browser. If the browser supports JavaScript, these controls will return a web page that incorporates JavaScript for a richer, more responsive user interface. However, no matter what the capabilities of the browser, *your* code is always executed on the server.



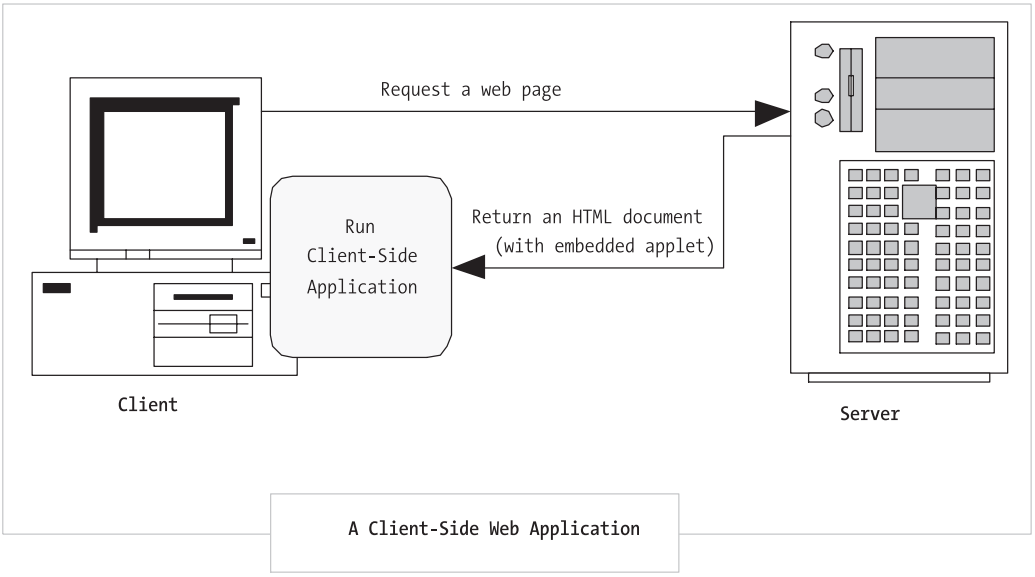


Figure 1-3. Server-side and client-side web applications

2786c01.fm Page 9 Saturday, December 20, 2003 6:30 PM

The Problems with ASP

The original ASP became more popular than even Microsoft had anticipated, and it wasn't long before it was being wedged into all sorts of unusual places, including mission-critical business applications and highly trafficked e-commerce sites. Because ASP hadn't been designed with these uses in mind, a number of problems began to appear. What began as a simple solution for creating interactive web pages became a complicated discipline that required knowledge in several fields as well as some painful experience.

If you've programmed with ASP before, you may already be familiar with some or all of these problems. They include the following:

- **Scripting limitations.** ASP applications rely on the VBScript language, which suffers from a number of limitations, including poor performance. To overcome these problems, developers usually need to add separately developed components, which add a new layer of complexity. In ASP.NET, web pages are designed in a modern .NET language, not a scripting language.

- **No application structure.** ASP code is inserted directly into a web page along with the HTML markup. The resulting tangle of code and HTML has nothing in common with today's modern, object-oriented languages. As a result, web form code can rarely be reused or modified without hours of effort.

- **Headaches with deployment and configuration.** Because of the way COM works, you can't easily update the components your website uses. Often, you need to manually stop and restart the server, which just isn't practical on a live website. Changing configuration options can be just as ugly. ASP.NET introduces a slew of new features to allow websites to be dynamically updated and reconfigured.

- **State limitations.** One of ASP's strongest features is its integrated session state facility, which allows you to retain temporary information about each client on the web server. However, session state is useless in scenarios where a website is hosted by several separate web servers. In this scenario, a client might access server B while its session information is trapped on server A and essentially abandoned. ASP.NET corrects this problem by allowing state to be stored in a central repository, such as a separate process or a database that all servers can access.

ASP.NET deals with these problems (and many more) by introducing a completely new model for web pages. This model is based on a remarkable new piece of technology called the .NET Framework.

The .NET Framework

The first thing that you should understand about the .NET Framework is that .NET is really a cluster of different technologies. These technologies include the following:

2786c01.fm Page 10 Saturday, December 20, 2003 6:30 PM

- **The .NET languages.** These include C# and Visual Basic .NET, the object-oriented and modernized successor to Visual Basic 6.0, as well as JScript .NET (a server-side version of JavaScript) and J# (a Java clone).
- **The CLR (Common Language Runtime).** The CLR is the engine that executes all .NET programs and provides automatic services for these applications, such as security checking, memory management, and optimization.
- **The .NET class library.** The class library collects thousands of pieces of prebuilt functionality that you can "snap in" to your applications. These features are sometimes organized into technology sets, such as ADO.NET (the technology for creating database applications) and Windows Forms (the technology for creating desktop user interfaces).
- **ASP.NET.** This is the engine that hosts web applications and web services, with almost any feature from the .NET class library. ASP.NET also includes a set of web-specific services.
- **Visual Studio .NET.** This optional development tool contains a rich set of productivity and debugging features. The Visual Studio .NET setup CD includes the complete .NET Framework.

The division between these components sometimes isn't clear. For example, ASP.NET is sometimes used in a very narrow sense to refer to the portion of the .NET class library used to design web pages. On the other hand, ASP.NET is also used to refer to the whole topic of .NET web applications, which includes .NET languages and many fundamental pieces of the class library that aren't web-specific. (That's generally the way that the term is used in this book. Our exhaustive examination of ASP.NET includes .NET basics, the VB .NET language, and topics that any .NET developer could use, such as component-based programming and database access.)

The .NET class library and CLR—the two fundamental parts of .NET—are shown in Figure 1-4.

The Confusion Behind .NET

Some developers have hesitated to embrace .NET because they believe they will be forced to subscribe to Microsoft's .NET Building Block Services (formerly called My Services and Hailstorm). These services, which are built on the .NET foundation, allow developers to use centralized user authentication, information, and notification services—for a cost. The debate over the validity of .NET Building Block Services is heated, but it really has little to do with ASP.NET development. You certainly won't need to use (or even understand) any part of the .NET Building Block Services product to create full-featured websites. In fact, at the time of this writing, version 1.1 of the .NET Framework is in widespread use, but the .NET Building Block Services still haven't been released, and their features still haven't been finalized.

2786c01.fm Page 11 Saturday, December 20, 2003 6:30 PM

Figure 1-4. The .NET Framework

2786c01.fm Page 12 Saturday, December 20, 2003 6:30 PM

VB .NET, C#, and the .NET Languages

Visual Basic .NET is a redesigned language that improves on traditional Visual Basic, and even breaks compatibility with existing Visual Basic 6 applications. Migrating to Visual Basic .NET is a stretch—and a process of discovery for most seasoned VB developers. The change is even more dramatic if you're an ASP or Office developer used to the scaled-down capabilities of VBScript.

C#, on the other hand, is an entirely new language. It resembles Java and C++ in syntax, but there is no direct migration path from Java or C++. In short, C#, like Visual Basic .NET, is an elegant, modern language ideal for creating the next generation of business applications.

Interestingly, C# and Visual Basic .NET are actually far more similar than Java and C# or Visual Basic 6 and VB .NET. Though the syntax is different, both C# and VB .NET use the .NET class library and are supported by the CLR. In fact, almost any block of C# code can be translated, line-by-line, into an equivalent block of VB .NET code. There is the occasional language difference (for example, VB .NET supports a language feature called optional parameters while C# doesn't), but for the most part, a developer who has learned one .NET language can move quickly and efficiently to another.

TIP The new languages present some challenges to even the most experienced of developers. In Chapters 3 and 4, you'll sort through the new syntax of Visual Basic .NET and learn the basics of object-oriented programming. By learning the fundamentals before you start creating simple web pages, you'll face less confusion and move more rapidly to advanced topics like database access and web services.

The Intermediate Language

All the .NET languages are compiled into another lower-level language before the code is executed. This lower-level language is the *MSIL* or just *IL* (Microsoft Intermediate Language). The CLR, the engine of .NET, only runs IL code. Because all .NET languages are designed based on IL, they all have profound similarities. This is the reason that the C# and VB .NET languages provide essentially the same features and performance. In fact, the languages are so compatible that a web page written with VB .NET can use a C# component in the exact same way it uses a VB .NET component, and vice versa.

Figure 1-5 shows how the .NET languages are compiled to IL. This diagram simplifies the compilation process slightly—in an ASP.NET application, a final machine-specific executable is created and cached on the web server to provide the best possible performance.

2786c01.fm Page 13 Saturday, December 20, 2003 6:30 PM

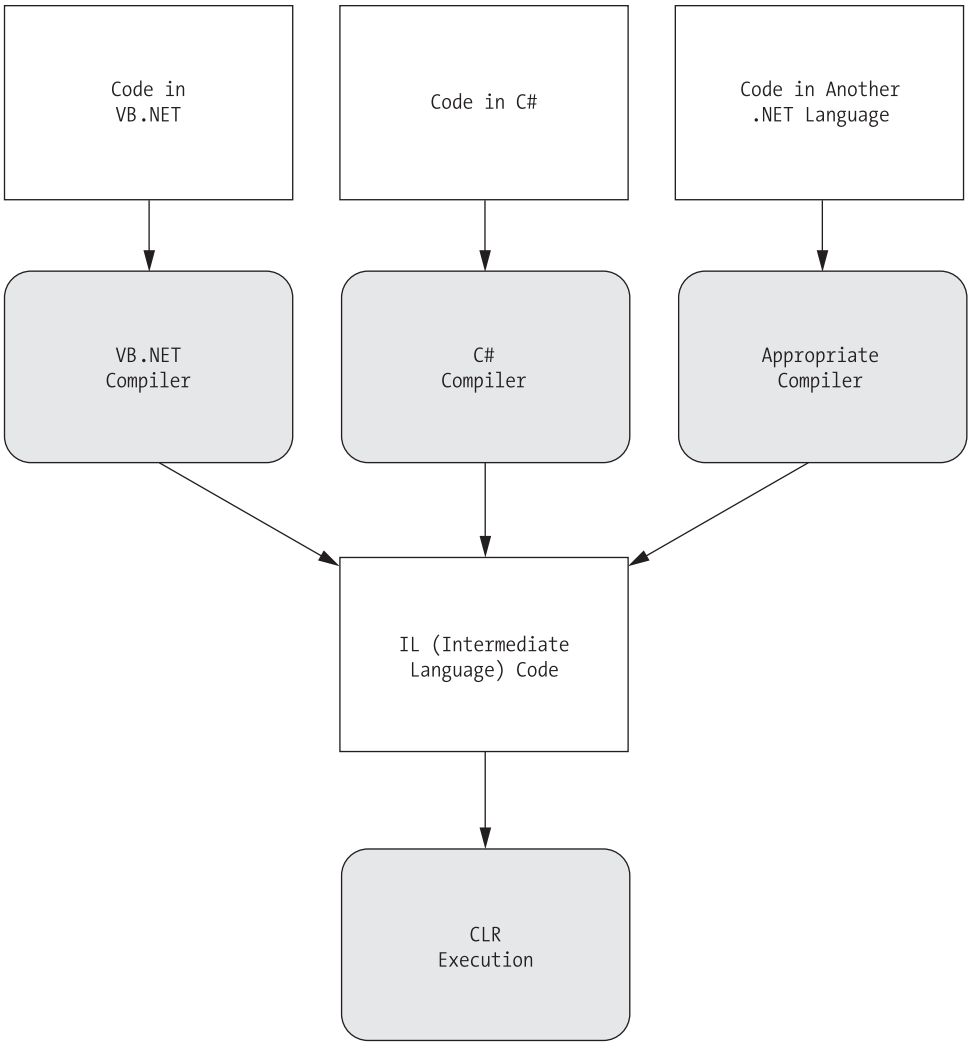


Figure 1-5. Language compilation in .NET

Other .NET Languages

C# and VB .NET aren't the only choices for ASP.NET development. Version 1.0 of the .NET Framework shipped with three languages: VB .NET, C#, and JScript .NET. In version 1.1, the .NET Framework added J#, a language with Java-like syntax. When creating an ASP.NET web application, you can use any of these languages. You can even use a .NET language provided by a third-party developer, such as a .NET version of Eiffel or Perl. This increasing range of language choices is possible because Microsoft has released a set of guidelines called the *CLS* (Common Language Specifications),

2786c01.fm Page 14 Saturday, December 20, 2003 6:30 PM

which define basic requirements and standards that allow other companies to write languages that can be compiled to IL.

This book focuses on Visual Basic .NET. Once you understand the fundamentals of the .NET Framework, you're free to explore other languages, although some (such as JScript .NET) may not support the full range of possible features.

The Common Language Runtime

The CLR is the engine that supports all the .NET languages. Many modern languages use runtimes. In Visual Basic 6, the runtime logic is contained in a DLL file named msvbvm60.dll. In C++, applications link to a file named mscrt40.dll. These runtimes may provide libraries used by the language, or they may have the additional responsibility of executing the code (as with Java).

Runtimes are nothing new, but the CLR represents a radical departure from Microsoft's previous strategy. For starters, the CLR and .NET Framework are much larger and more ambitious than the Visual Basic 6 or C++ runtimes. The CLR also provides a whole set of related services such as code verification, optimization, and garbage collection.

NOTE The CLR is the reason that some developers have accused .NET of being a Java clone. The claim is fairly silly. It's true that .NET is quite similar to Java in key respects (both use a special managed environment and provide features through a rich class library), but it's also true that every programming language "steals" and improves from previous programming languages. This includes Java, which adopted parts of the C/C++ language and syntax when it was created. Of course, in many other aspects .NET differs just as radically from Java as it does from VBScript.

All .NET code runs inside the CLR. This is true whether you're running a Windows application or a web service. For example, when a client requests an ASP.NET web page, the ASP.NET service runs inside the CLR environment, executes your code, and creates a final HTML page to send to the client.

The implications of the CLR are wide-ranging:

- **Deep language integration.** C# and VB .NET, like all .NET languages, compile to a special language called IL. In other words, the CLR makes no distinction between different languages—in fact, it has no way of knowing what language was used to create an executable. This is far more than mere language compatibility; it's language *integration*.

2786c01.fm Page 15 Saturday, December 20, 2003 6:30 PM

- **Side-by-side execution.** The CLR also has the ability to load more than one version of a component at a time. In other words, you can update a component many times, and the correct version will be loaded and used for each application. As a side effect, multiple versions of the .NET Framework can be installed, meaning that you'll be able to upgrade to new versions of ASP.NET without replacing the current version or needing to rewrite your applications.

- **Fewer errors.** Whole categories of errors are impossible with the CLR. For example, the CLR prevents the wide variety of memory mistakes that are possible with lower-level languages like C++. For Visual Basic programmers, many of these advantages are nothing new—they're similar to features of the traditional VB environment.

Along with these truly revolutionary benefits, there are some other potential drawbacks. The following are three issues that are often raised by new developers, but aren't always answered:

- **Performance.** A typical ASP.NET application is much faster than a comparable ASP application, because ASP.NET code is compiled natively. However, other .NET applications probably won't match the blinding speed of well-written C++ code, because the CLR imposes some additional overhead. Generally, this is only a factor in a few performance-critical high-workload applications (like real-time games). With high-volume web applications, the potential bottlenecks are rarely processor-related, but are usually tied to the speed of an external resource such as a database or the web server's file system. With ASP.NET caching and some well-written database code, you can ensure excellent performance for any web application.

- **Code transparency.** The IL language is much easier to disassemble, meaning that if you distribute a compiled application or component, other programmers may have an easier time determining how your code works. This isn't much of an issue for ASP.NET applications, which aren't distributed, but are hosted on a secure web server.

- **Cross-platform support?** No one is entirely sure whether .NET will be adopted for use on other operating systems and platforms. Ambitious projects, like Mono (.NET on Linux) are currently underway (see `http://www.go-mono.com`). However, .NET will probably never have the wide reach of a language like Java because it incorporates too many different platform-specific and operating system–specific technologies and features.

The .NET Class Library

The .NET class library is a giant repository of classes that provide prefabricated functionality for everything from reading an XML file to sending an email message. If you've had any exposure to Java, you may already be familiar with the idea of a class

2786c01.fm Page 16 Saturday, December 20, 2003 6:30 PM

library. However, the .NET class library is more ambitious and comprehensive than just about any other programming framework. Any .NET language can use the .NET class library's features by interacting with the right objects. This helps encourage consistency among different .NET languages and removes the need to install numerous separate components on your computer or web server.

Some parts of the class library include features you'll never need to use in web applications (such as the classes used to create desktop applications with the Windows interface). Other parts of the class library are targeted directly at web development, like those used for web services and web pages. Still more classes can be used in a number of different programming scenarios, and aren't specific to web or Windows development. These include the base set of classes that define common variable types, and the classes for data access, to name just a few.

You'll explore the .NET Framework throughout this book. In the meantime, here are some general characteristics of the .NET Framework:

- **Open standards.** Microsoft currently provides programming tools that allow you to work with many open standards, such as XML. In .NET, however, many of these standards are "baked in" to the framework. For example, ADO.NET (Microsoft's data access technology) uses XML natively, behind the scenes. Similarly, web services work automatically through XML and HTTP. This deep integration of open standards makes cross-platform work much easier.

- **Emphasis on infrastructure.** Microsoft's philosophy is that they will provide the tedious infrastructure so that application developers only need to write business-specific code. For example, the .NET Framework automatically handles files, databases, and transactions. You just add the logic needed for your specific application.

- **Disconnected model.** The .NET Framework emphasizes distributed and Internet applications. Technologies like ADO.NET are designed from the ground up to be scalable even when used by hundreds or thousands of simultaneous users.

Visual Studio .NET

The last part of .NET is the optional Visual Studio .NET editor, which provides a rich environment where you can rapidly create advanced applications. You don't need to use Visual Studio .NET to create web applications. In fact, you might be tempted to use the freely downloadable .NET Framework and a simple text editor to create ASP.NET web pages and web services. However, in doing so you'll multiply your work, and you'll have a much harder time debugging, organizing, and maintaining your code.

Some of the features of Visual Studio .NET include the following:

- **Page design.** You can create an attractive page with drag-and-drop ease using Visual Studio .NET's integrated web form designer. There's no need to understand HTML.

2786c01.fm Page 17 Saturday, December 20, 2003 6:30 PM

- **Automatic error detection.** You could save hours of work when Visual Studio .NET detects and reports an error before you run your application. Potential problems are underlined, just like the "spell-as-you-go" feature found in many word processors.
- **Debugging tools.** Visual Studio .NET retains its legendary debugging tools, which allow you to watch your code in action and track the contents of variables.
- **IntelliSense.** Visual Studio .NET provides statement completion for recognized objects, and automatically lists information such as function parameters in helpful tooltips.

I recommend Visual Studio .NET highly. However, in this book you'll start with the basics of ASP.NET, code behind, and web-control development using the Notepad text editor. This is the best way to get a real sense of what is happening behind the scenes. Once you understand how ASP.NET works, you can quickly master Visual Studio .NET, which is introduced in Chapter 8. I hope you'll use Visual Studio .NET for all the remaining examples, but if you prefer Notepad or a third-party tool like the freely downloadable Web Matrix, you can use that instead—with a little more typing required.

The Last Word

This chapter presented a high-level overview that gave you your first taste of some of the radical changes that will affect ASP.NET applications as well as some of the long-awaited improvements that are finally in place. But before you can get to work writing web pages, you need to continue with Chapter 2, which shows you how to prepare your computer with the Internet Information Services web hosting software.

2786c01.fm Page 18 Saturday, December 20, 2003 6:30 PM