



1

Getting Started with ASP.NET

ASP.NET is a new and powerful technology for creating dynamic web pages. It's a convergence of two major Microsoft technologies, Active Server Pages (ASP) and .NET. ASP is a relative old-timer on the web computing circuit and has provided a sturdy, powerful, and effective way of creating dynamic web pages for five years or so now. .NET is the new kid on the block and is a whole suite of technologies designed by Microsoft with the aim of revolutionizing the way in which programming development is conducted in the future, and the way companies carry out business. As a conjunction of the two, ASP.NET is a way of creating dynamic web pages while making use of the innovations present in .NET.

The first important thing to know about ASP.NET is that you don't need any ASP skills to be able to learn it. In fact all you need is a little HTML knowledge for building your own web pages. Knowing any ASP could in some ways be a disadvantage, because you might have to 'unlearn' some of the principles you previously held to be true. ASP.NET is a more powerful technology than its old namesake. Not only does it allow you to build dynamic web pages, but it also tailors the output in HTML to whatever browser you're using, and comes with a great set of reusable, predefined, and ready to use controls for use in your ASP.NET projects, which reduce the amount of code you have to write, so you can be more productive while programming.

So what can you do with ASP.NET? It might be easier to list what you can't, as that is arguably shorter! One of the most eye-catching things is the way you can use any programming language based in .NET, such as C#, VB .NET, or JScript .NET, to create your web applications. Within these applications, ASP.NET enables you to customize pages for a particular user, makes it much simpler now to keep track of a particular user's details as they move around, and makes storing information to a database or self-describing XML document faster and easier. You can alter the layout of the page using a visual IDE rather than having to figure out positioning within code, and even alter the contents of files on your machine (if you have the correct permissions). You can also use bits and pieces of other applications without downloading the whole application: for example, you can access a zip code verifier that is exposed by another web site, without having to download the whole of that application, or giving your users the impression that they've left your site (we'll talk more about this in the *Web Services* chapter later on). Basically, with ASP.NET the applications that you create are only limited by your imagination.

Chapter 1

In this first chapter we'll be mainly concerned with ASP.NET's installation process. We'll start with a quick introduction to the world of web servers, dynamic web pages, and a little bit about what ASP.NET is, but what we really aim to achieve is to get you up and running a fully functional web server, with a fully functional ASP.NET installation. By the end of the chapter you'll have created a short ASP.NET test page to check that both the web server and ASP.NET are both working as intended. We'll also have a look at some of the most common pitfalls encountered, just in case things don't go as planned!

We will cover the following topics:

- ❑ Static Web Pages
- ❑ Dynamic Web Pages
- ❑ An overview of the different technologies for creating dynamic web pages, including ASP.NET
- ❑ Installing Internet Information Services (IIS)
- ❑ Installing the .NET Framework
- ❑ Testing and troubleshooting your installation

What is a Static Web Page?

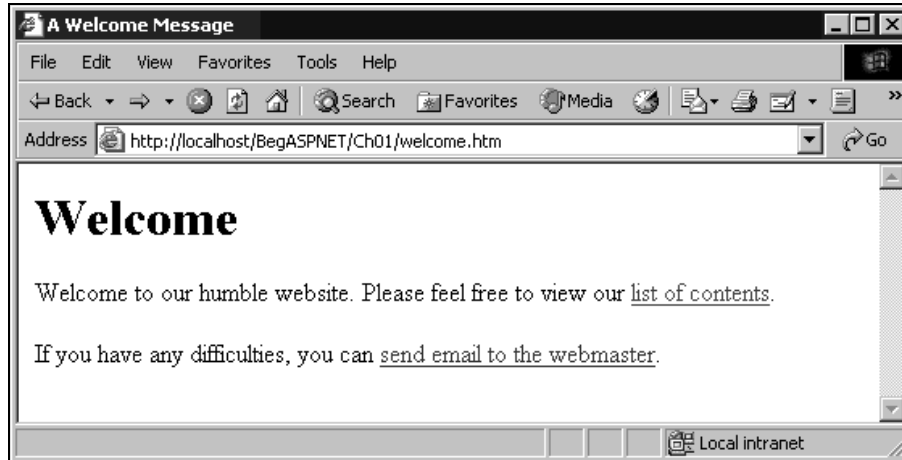
If you surf around the Internet today, you'll see that there are lots of static web pages out there. What do we mean by a **static** web page? Essentially, it's a page whose content consists of some HTML code that was typed directly into a text editor and saved as an `.htm` or `.html` file. Thus, the author of the page has already determined the *exact* content of the page, in HTML, at some time before any user visits the page.

Static web pages are often quite easy to spot: sometimes you can pick them out by just looking at the content of the page. The content (text, images, hyperlinks, etc.) and appearance of a static web page is *always* the same – regardless of *who* visits the page, or *when* they visit, or *how* they arrive at the page, or any other factors.

For example, suppose we create a page called `welcome.htm` for our web site, by writing some simple HTML like this:

```
<html>
<head><title>A Welcome Message</title></head>
<body>
  <h1>Welcome</h1>
  Welcome to our humble website. Please feel free to view our
  <a href="contents.htm">list of contents</a>.
  <br><br>
  If you have any difficulties, you can
  <a href="mailto:webmaster@wrox.com">send email to the webmaster</a>.
</body>
</html>
```

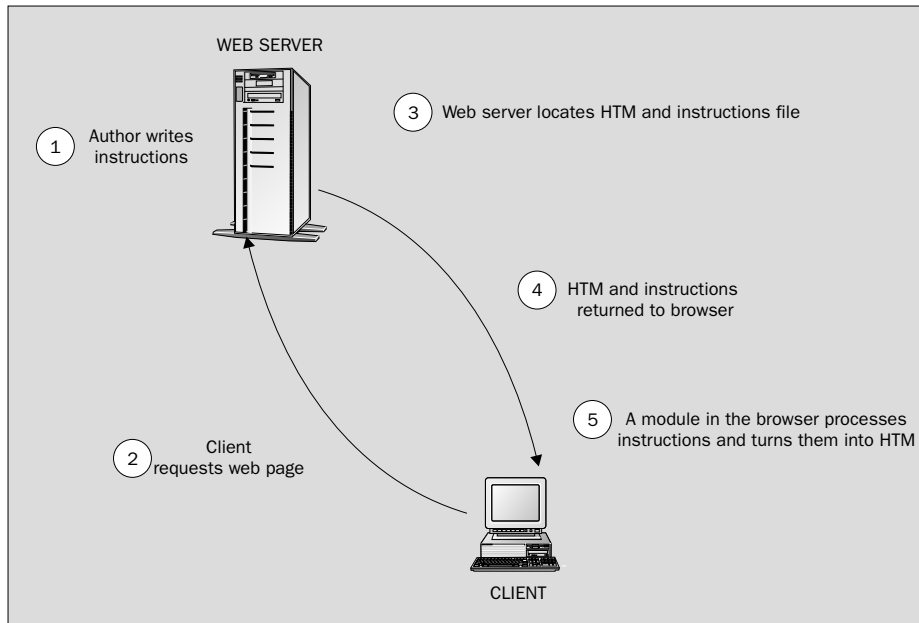
Whenever any client comes to our site to view this page, it will look like this. The content of the page was determined *before* the request was made – at the time the Webmaster saved the .htm file to disk:



How are Static Web Pages Served?

OK, so let's think for a moment about how a static, pure-HTML page finds its way onto a client browser:

- 1.** A web author writes a page composed of pure HTML, and saves it within an .htm file on the server.
- 2.** Sometime later, a user types a page request into their browser, and the request is passed from the browser to the **web server**.
- 3.** The web server locates the .htm page and converts it to an HTML stream.
- 4.** The web server sends the HTML stream back across the network to the browser.
- 5.** The browser processes the HTML and displays the page.



Static, pure-HTML files like `Welcome.htm` make perfectly serviceable web pages. We can even spruce up the presentation and usability of such pages by adding more HTML to alter the fonts and colors. However, there's only so much we can achieve by writing pure HTML, precisely because the content is completely determined *before* the page is ever requested.

The Limitations of Static Web Pages

For example, suppose we want to enhance our `Welcome` page – so that it displays the current time or a special message that is personalized for each user. These are simple ambitions, but they are impossible to achieve using HTML alone. If you're not convinced, try writing a piece of HTML for a web page that displays the current time, like this:



As you type in the HTML, you'll soon realize the problem – you know that the user will request the page sometime, but you don't know *what the time will be* when they do so! Hard-coding the time into your HTML will result in a page that always claims that the time is the same (and will almost always display the wrong time).

In other words, you're trying to write pure HTML for a web page that displays the time – but you can't be sure of the *exact* time that the web page should display until the time the page is requested. It can't be done using HTML alone.

Also, HTML offers no features for personalizing your web pages: each web page that is served is the same for every user. There's also no security with HTML: the code is there for everybody to view, and there's nothing to stop you from copying somebody else's HTML code and using it in your own web page. Static pages might be very fast, as quick as copying a small file over a network, but they are quite limited without any dynamic features.

Since we can't create our page by saving our hard-coded HTML into a file *before* the page is requested, what we need is a way to generate the HTML *after* the page is requested. There are two ways of doing this. We'll look at them both shortly, but before we go any further we need to make sure everybody is up to speed on the terminology we've introduced here.

What is a Web Server?

A **web server** is a piece of software that manages web pages and makes them available to 'client' browsers – via a local network or over the Internet. In the case of the Internet, the web server and browser are usually on two different machines, possibly many miles apart. However, in a more local situation we might set up a machine that runs the web server software, and then use a browser on the *same* machine to look at its web pages. It makes no difference whether you access a remote web server (that is, a web server on a different machine from your browser application) or a local one (web server and browser on the same machine), since the web server's function – to make web pages available to all – remains unchanged. It might well be that you are the only person with access to your web server on your own machine, as would be case if you were running a web server from your home machine. Nevertheless, the principles remain the same.

While there are many web servers available – the commonest ones being Apache, Internet Information Services (or IIS, for short) and Iplanet's Enterprise server – we're only going to talk about one in this book: Microsoft's IIS. This is because it is the only web server that will run ASP.NET. The web server comes as part of the installation for both Windows 2000 and Windows XP. IIS version 5.0 comes with Windows 2000 and IIS version 5.1 with Windows XP; however, there is very little to distinguish the two, and we shall treat them in this chapter as the same product. We'll look at how you go about installing IIS shortly, but first, let's take a look at its role in helping to create dynamic web pages.

How are Dynamic Web Pages Served?

To fully understand the nature of dynamic web pages, we first need to look at the limitations of what you can and can't do with a static web page.

Two Ways of Providing Dynamic Web Page Content

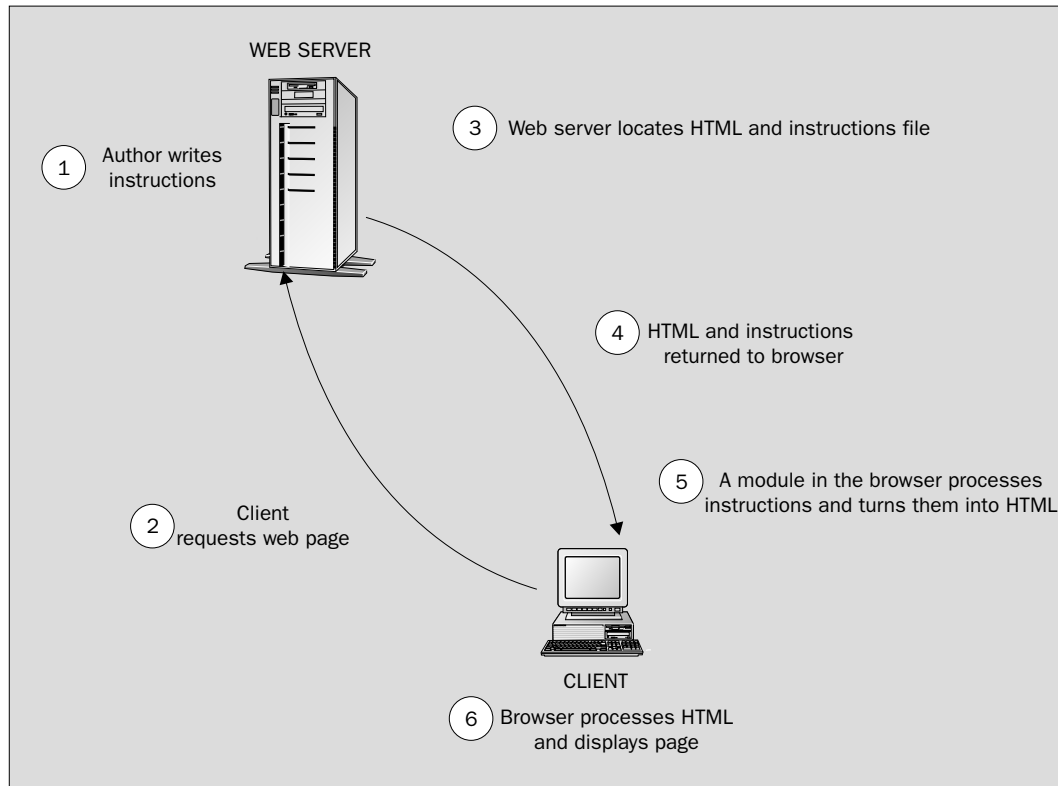
Even though, in this book, we're only going to be creating dynamic web pages using one of two very different methods, we need to be aware of the differences between these two different ways of doing things, as the underlying principles for both feature heavily throughout the book.

Client-Side Dynamic Web Pages

In the client-side model, modules (or plug-ins) attached to the browser do all the work of creating dynamic pages. The HTML code is typically sent to the browser, along with a separate file containing a set of **instructions**, which is referenced from within the HTML page. However, it is also quite common to find these instructions intermingled with the HTML code. The browser then uses them to generate pure HTML for the page when the user requests the page – in other words, the page is generated **dynamically** on request. This produces an HTML page, which is displayed in the browser.

So in this model our set of five steps now becomes six:

- 1.** A web author writes a set of instructions for creating HTML, and saves it within an `.htm` file. The author also writes a set of instructions in a different language. This might be contained within the `.htm` file, or within a separate file.
- 2.** Sometime later, a user types a page request into their browser, and the request is passed from the browser to the web server.
- 3.** The web server locates the `.htm` page, and may also have to locate a second file that contains the instructions.
- 4.** The web server sends both the newly created HTML stream and instructions back across the network to the browser.
- 5.** A module within the browser processes the instructions and returns the results as HTML within the `.htm` page – only one page is returned, even if two were requested.
- 6.** The HTML is then processed by the browser, which displays the page.



Client-side technologies have fallen out of favor in recent times, as they take a long time to download, especially if you have to download more than one file. A second drawback is that each browser interprets client-side scripting code in different ways, so you have no way of guaranteeing that if Internet Explorer understands them, Netscape Navigator or Opera will also be able to process them. Other major drawbacks are that it is a problem to write client-side code that uses server-side resources such as databases because it is interpreted at client-side. Also, client-side scripting code isn't secure and can be easily viewed with the View Source Code option on any browser, which might also be undesirable, and may compromise the security of the web site.

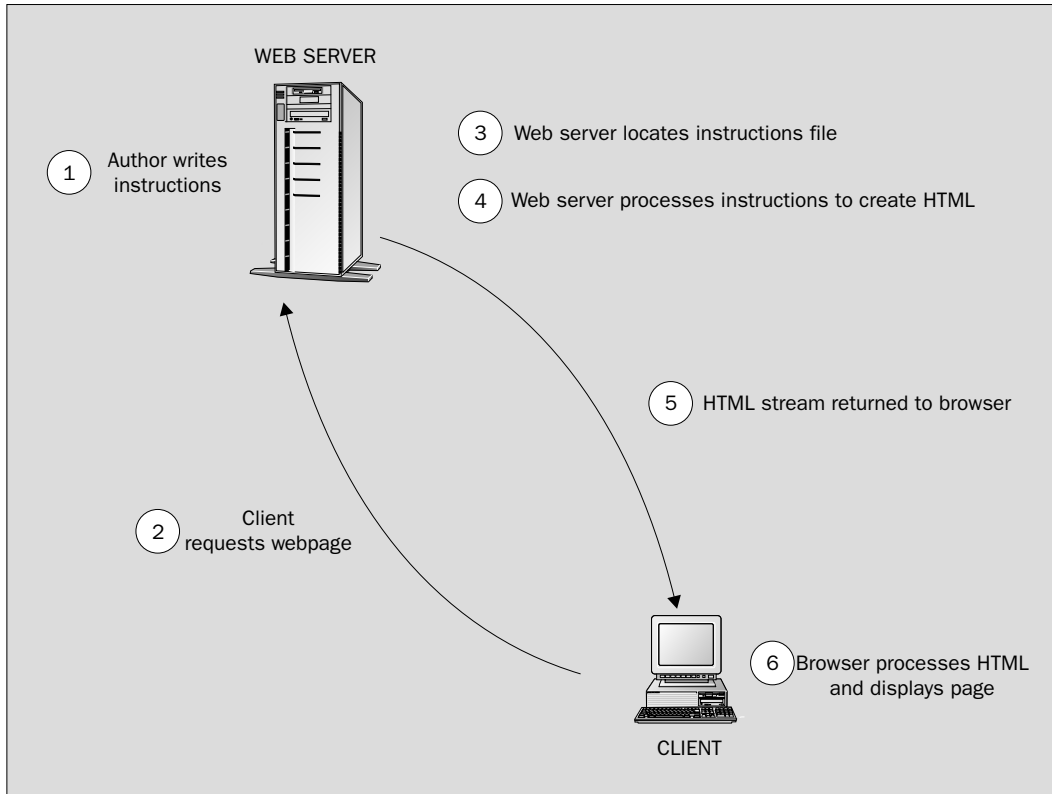
Server-Side Dynamic Web Pages

With the server-side model, the HTML source is sent to the web server with an intermingled set of **instructions**. Again this set of instructions will be used to generate HTML for the page at the time the user requests the page. Once again, the page is generated dynamically upon request. However, there is a subtle twist regarding where the processing of instructions is done:

1. A web author writes a set of instructions for creating HTML, and saves these instructions within a file.
2. Sometime later, a user types a page request into their browser, and the request is passed from the browser to the web server.

Chapter 1

3. The web server locates the file of instructions.
4. The web server follows the instructions in order to create a stream of HTML.
5. The web server sends the newly created HTML stream back across the network to the browser.
6. The browser processes the HTML and displays the page.



The twist is that all the processing is done on the server, before the page is sent back to the browser. One of the key advantages this has over the client-side model is that only the HTML is actually sent to the browser. This means that our page's original code is hidden away on the server, and that we can safely assume that most browsers should be able to at least have a go at displaying the generated HTML. ASP.NET, as you might have gathered, does its processing on the server-side.

While both processes for serving a dynamic web page add only a single step to the process for serving a static web page (Step 5 on the client or Step 4 on the server) this single step is crucial. In this step the HTML that defines the web page is not generated until *after* the web page has been requested. For example, we can use either technique to write a set of instructions for creating a page that displays the current time:


```
<html>
<head><title>The Punctual Web Server</title></head>
<body>
  <h1>Welcome</h1>
  In Webserverland, the time is exactly
  <INSTRUCTION: produce HTML to display the current time>
</body>
</html>
```

In this case, we can compose most of the page using pure HTML. It's just that we can't hardcode the current time. Instead, we can write special code (which would replace the highlighted line here) that instructs the web server to generate that bit of HTML – during Step 5, on the client, or Step 4, on the server – at the time the page is requested. We'll return to this example later in the chapter, and we'll see how to write the highlighted instruction using ASP.NET.

Now we're going to look at the various different technologies, including ASP.NET, and see how the logic is supported in each.

An Overview of the Technologies

You've just seen that there are also two distinct models for providing dynamic content. ASP.NET falls into the server-side model. However, we're going to look at what we consider to be the most important technologies in both models, as we will reference some of the client-side models in later chapters, particularly if we mention old-style ASP. Not all of the technologies work in the same way as ASP.NET, but they all allow the user to achieve the same endresult – that of dynamic web applications. If ASP.NET is not an ideal solution to your problems, then you might want to consider these following technologies, taking into account the following questions:

- Are they supported on the platform you use?
- Are they difficult to learn?
- Are they easy to maintain?
- Do they have a long-term future?
- Do they have extra capabilities, such as being able to parse XML?
- Are a lot of people already using them – are there a lot of tools available?
- Are the support, skills, and knowledge required to use them readily available?

We're now going to give a quick overview of what each one does, and in doing so, try to give you an idea of where ASP.NET (and the ASP technology that preceded it) fits in to the big picture.

Client-Side Technologies for Providing Dynamic Content

Each of these technologies relies on a module (or plug-in) built into the browser to process the instructions we talked about earlier. The client-side technologies are a mishmash of scripting languages, controls, and fully fledged programming languages.

JavaScript

JavaScript is the original browser scripting language, and is not to be confused with Java. Java is a complete application programming language in its own right. Netscape originally developed a scripting language, known as LiveScript, to add interactivity to its web server and browser range. It was introduced in the release of the Netscape 2 browser, when Netscape joined forces with Sun, and in the process, they changed its name to **JavaScript**. JavaScript borrows some of its syntax and basic structures from Java (which in turn borrowed ideas from C), but has a different purpose – and evolved from different origins (LiveScript was developed separately from Java).

For example, while JavaScript can control browser behavior and content, it isn't capable of controlling features such as file handling. In fact JavaScript is actively prevented from doing this for security reasons. Think about it: you wouldn't want a web page capable of deleting files on your hard drive, would you? Meanwhile, Java can't control the browser as a whole, but it can do graphics and perform network and threading functions.

JavaScript is much simpler to learn than Java. It is designed to create small, efficient applications that can do many things, from performing repetitive tasks to handling events generated by the user (such as mouse clicks, keyboard responses, and so on). However, JavaScript's functionality is necessarily restricted.

Microsoft introduced their own version of JavaScript, known as **JScript**, in Internet Explorer 3.0 and have supported it ever since, right up to and including IE6. It has only minor differences from the Netscape version of the language, although in older versions of both browsers the differences were more considerable.

VBScript

In Internet Explorer 3.0, Microsoft also introduced its own scripting language, VBScript, which was based on its Visual Basic programming language. VBScript was intended to be a direct competitor to JavaScript. In terms of functionality, there isn't much difference between the two: it's more a matter of personal preference – VBScript has a similarly reduced functionality. Visual Basic developers sometimes prefer VBScript because VBScript is, for the most part, a subset of Microsoft's Visual Basic language (the forerunner of VB.NET). However, it enjoys one advantage that makes it more attractive to novice programmers, in that unlike JavaScript, it isn't case-sensitive and is therefore less fussy about the particulars of the code. Although this feature has the drawback that this makes it a lot slower and less efficient.

However, the biggest drawback by far is that there isn't a single non-Microsoft browser that supports VBScript for client-side scripting. For a short while there were some proprietary plug-ins for Netscape that provided VBScript support, but these never took off. You'll find that JavaScript is much more widely used and supported. If you want to do client-side scripting of web pages on the Internet then JavaScript is the only language of choice. VBScript should only be considered when working on intranet pages where it is known that all clients are IE on Windows.

With both JavaScript and VBScript there is a module, known as a script engine, built into the browser that dynamically processes the instructions, or script, as it is known in this case.

Java Applets

Java is a cross-platform language for developing applications. When Java first hit the Web in the mid-1990s, it created a tremendous stir. The idea is to use Java code in the form of **applets**, which are essentially Java components that can be easily inserted into web pages with the aid of the `<applet>` tag.

Java enjoys better functionality than scripting languages, offering better capabilities in areas such as graphic functions and file handling. Java is able to provide these powerful features without compromising security because the applets run in what is known as a sandbox – which prevents a malicious program downloaded from the Web from doing damage to your system. Java also boasts strong database support through JDBC (a technology for connecting to data sources).

Microsoft and Netscape browsers both have built-in Java support via something known as the Java Virtual Machine (JVM), and there are several standard `<object>` and non-standard `<applet>` tags that are used to add Java applets to a web page. These tags tell the browser to download a Java file from a server and execute it with the Java Virtual Machine built into the browser. Of course, this extra step in the web page building phase means that Java applets can take a little while to download, and can take even longer to process once on the browser. So while smaller Java applets (that provide features such as dropdown menus and animations) are very popular on the Web, larger ones are still not as widespread as scripted pages.

Although the popularity of Java today isn't quite what some people expected, it makes an ideal teaching tool for people wishing to break into more complex languages, and its versatility makes it well suited for programming web applications.

Flash

Flash from Macromedia is a web motion graphics tool, enabling developers to create animations, interactive graphics, and user interface elements such as menus. While this might seem slightly at odds with the other fully fledged scripting and programming languages discussed here, Flash actually provides its own scripting language, ActionScript, and as a result is fast becoming the standard way of providing client-side dynamic content. It is also fully interoperable with many server-side technologies.

A plug-in enabling you to use and view Flash web pages comes as standard with most modern versions of Internet Explorer and Netscape Navigator, but can be downloaded from the www.macromedia.com if not present. The Flash tool that must be used to create Flash animations (`.swf` files) must be purchased from Macromedia, although a 30 day demo version is available for free.

Such is the power and versatility of Flash that it has become a popular catch-all for developers who wish to include graphics or sound. The Actionscript language it uses resembles JavaScript in many ways, which is an added advantage for developers already familiar with JavaScript. Flash files are added to HTML pages using the Flash tool. In reality the tool auto-generates an `<object>` or `<embed>` tag, which the browser recognizes and deals with appropriately.

Server-Side Technologies for Providing Dynamic Content

The server-side technologies rely on modular attachments added onto the web server rather than the browser. Consequently, only HTML, and any client-side script, is sent back to the browser by the web server. In other words, none of the server-side code is sent back. Server-side technologies have a more consistent look and feel than client-side ones, and it doesn't take that much extra learning to move between some of the server-side technologies (excepting CGI).

CGI

The **Common Gateway Interface (CGI)** is a mechanism for creating scripts on the server, which can then be used to create dynamic web applications. CGI is a module that is added to the web server. It has been around for quite a bit longer than even ASP, and right now a large proportion of dynamically created web pages are created using CGI and a scripting language. However, it's incorrect to assume that CGI does the same job as ASP.NET or ASP. Rather, CGI allows the user to invoke another program (such as a Perl script) on the web server in order to create the dynamic web page, and the role of CGI is to pass data – which might be supplied by the user – to the this program for processing. However, it does provide the same end result – a dynamic web application.

However, CGI has some severe shortcomings:

- ❑ It is not easy for a beginner to learn how to program such modules
- ❑ CGI requires a lot of server resources, especially in a multi-user situation
- ❑ It adds an extra step to our server-side model of creating dynamic content: namely, it's necessary to run a CGI program to create the dynamic page, before the page is processed on the server

What's more, the format in which CGI receives and transmits data means that the data cannot be handled in any straightforward manner by most programming languages: you need one with good facilities for manipulating text and communicating with other software. The most suitable programming languages that can work on any operating system for doing this are C, C++, and Perl. While they can more than adequately do the job for you, they're some of the more complex languages to learn. Visual Basic doesn't offer enough text handling facilities, and is therefore rarely used with CGI.

Despite this, CGI is still very popular with many big web sites, particularly those running on UNIX operating systems. It also runs on many different platforms, which will ensure its continued popularity.

ASP

Active Server Pages (ASP) is now dubbed 'Classic ASP', and if you see this term in the book, we will be using it to describe any ASP that isn't ASP.NET. ASP commonly relied on either of the JavaScript or VBScript scripting languages (although it was also possible to use any scripting language installed on Windows, such as PerlScript) to create dynamic web pages. ASP is a module (the `asp.dll` file) that you attach to your web server, and which then processes the JavaScript/VBScript on the web server, producing HTML, which is then sent, via the server, to the client. Processing is thus all done on the server, rather than the browser.

ASP lets us use practically any of the functionality provided by Windows, such as database access, e-mailing, graphics, networking, and system functions, and all from within a typical ASP page. However, ASP's shortcomings are that it is very, very slow. It is also restricted to using only scripting languages: it can't do all the things that a fully-fledged programming language can. Secondly, the scripting languages, being the 'junior' versions of full programming languages, took a lot of shortcuts to make the language smaller. Some of these shortcuts mean that our programs written with scripting languages are longer and more complicated than is otherwise necessary. As we're going to see, ASP.NET rectifies a lot of this by making code more structured, easier to understand, and shorter.

JSP

JavaServer Pages (JSP) is a technology that enables us to combine markup (HTML or XML) with Java code to dynamically generate web pages. The JSP specification is implemented by several web servers, in contrast with ASP, which is only supported under IIS, and plug-ins are available that enable us to use JSP with IIS 4.0/5.x. One of the main advantages of JSP is the portability of code between different servers. JSP is also very powerful, faster than ASP, and instantly familiar to Java programmers. It allows the Java program to leverage the aspects of the Java2 platform such as JavaBeans and the Java 2 libraries. JavaServer Pages isn't directly related to ASP, but it does boast the ability to embed Java code into your web pages using server-side tags. More details about JSP can be found at the official site at www.javasoft.com/products/jsp/index.html and at the JSP FAQ at www.esperanto.org.nz/jsp/jspfaq.html.

ColdFusion

ColdFusion (www.macromedia.com/software/coldfusion/) also enables servers to access data as the server builds an HTML page. Cold Fusion is a module installed onto your web server. Like ASP, ColdFusion pages are readable by any browser. ColdFusion also utilizes a proprietary set of tags, which are processed by the ColdFusion Server software. This server software can run on multiple platforms, including IIS, Netscape Enterprise Server, and Unix/Apache. The major difference is that while ASP.NET solutions are built primarily with programming languages and objects, ColdFusion utilizes HTML-like tags, which encapsulate functionality. A drawback is that the ColdFusion software doesn't come for free, and indeed you could find yourself paying well in excess of a thousand dollars for the privilege of running Cold Fusion on your web server.

PHP

PHP (originally **Personal Home Pages**, but more recently **PHP Hypertext Preprocessor**) is another scripting language for creating dynamic web pages. When a visitor opens the page, the server processes the PHP commands and then sends the results to the visitor's browser, just as with ASP.NET or ColdFusion. Unlike ASP.NET or ColdFusion, however, PHP is open-source and cross-platform. PHP runs on Windows NT and many Unix versions, and runs on the Apache web server (the free web server that commonly runs on UNIX and other related platforms). When built as an Apache module, PHP is especially speedy. A downside is that you have to download PHP separately and go through a series of quite complex steps to install it and get it working on your machine. Also, PHP's session management (the management of numerous simultaneous users) was non-existent until PHP 4, and is still inferior to ASP's even now.

PHP's language syntax is similar to C and Perl. This might prove a barrier to people with no prior programming experience, but if you have a background in either language then you might want to take a look. PHP also has some rudimentary object-oriented features, providing a helpful way to organize and encapsulate your code. You can find more information about PHP at www.php.net.

ASP.NET

So why are we looking at all these other technologies if in this book we're only going to be learning about ASP.NET? Hopefully, you'll see a similarity between the technologies, and this will aid your understanding of ASP.NET.

ASP.NET also relies on a module attached to the web server. However, the ASP.NET module (which is a physical file called `aspnet_isapi.dll`) doesn't do all of the work itself: it passes some on to the .NET Framework to do the processing for it. Rather than going into ASP.NET in this subsection here, it's time to start talking about it as a separate entity in its own right, as this is the focus of the book.

What is ASP.NET?

We're going to be asking this question a lot throughout the book, and each time we ask it, we're going to give you a slightly more in-depth answer. If we were to give you a full answer now, you'd be overwhelmed by as-yet meaningless jargon. So, you'll probably be aware of some unanswered questions each time we describe it.

Our original definition, right at the very start of the chapter, was "ASP.NET is a new and powerful technology for creating dynamic web pages", and this still holds true. However, as you now know, it isn't the only way to deliver dynamic web pages, so let's refine our definition a little to read:

ASP.NET is a new and powerful server-side technology for creating dynamic web pages.

Secondly, ASP.NET isn't the only thing that we're interested in. In fact, it's one of a set of technologies that comprise the **Microsoft .NET Framework**. For now, you can think of this as a giant toolkit for creating all sorts of applications, and in particular, for creating applications on the Web. When we come to install ASP.NET we will also be installing the .NET Framework at the same time, and we'll be using bits and pieces of the .NET Framework throughout the book.

How does ASP.NET Differ from ASP?

Steady on! We're just getting to this part. As we've already said, ASP is restricted to using scripting languages, mainly JavaScript or VBScript (although it can use any scripting language supported by the Windows system). We add ASP code to our pages in the same way as we do client-side script, and this leads to problems such as messy coding and restricted functionality. ASP.NET has no such problems.

First off, ASP.NET enables you to use a far greater selection of full programming languages, and also enables you to utilize to the full the rich potential of the .NET Framework. It helps you create faster, more reliable dynamic web pages with any of the programming languages supported by the .NET Framework. Typical languages supported natively are C#, VB .NET, and a new version of JScript called JScript .NET. On top of this, it is expected that third-party developers will create versions of Perl, Python and many others to work in ASP.NET. And no, before you ask, we don't expect you to know any of these programming languages. We're going to choose one language, C#, and teach you ASP.NET with it. We've chosen C#, as it's arguably the most popular of the .NET languages, and it can do pretty much anything that the other languages we mentioned can do. Lastly, and most importantly, we've chosen C# as it comes free with ASP.NET – so when you install ASP.NET, you get C# as well.

At this stage you might be thinking, "Hang on, I've got to figure out C#, then I've got to get a handle on ASP.NET – that sounds like an awful lot to learn." Don't worry; you won't be learning two languages. ASP.NET, as we said right from the beginning, is not a language – it is a technology. This technology is accessible via a programming language. What we're going to be doing is teaching you ASP.NET features as we teach you C#. So in other words, you will be creating your web pages using C# and using ASP.NET to drive it. However, before you rush out and get a C# book instead, we will be approaching the language from the angle of creating dynamic web pages only.

In summation, ASP.NET is a server-side technology that lets you use fully-fledged programming languages to create your web pages.

I'm Still Confused about ASP, ASP.NET, and C#

It's really important to get these terms separate and distinct in your mind, so before we move on to actually installing and running ASP.NET, we're going to go back and redefine them just to make sure:

- ❑ **ASP** – a server-side technology for creating dynamic web pages that only lets you use scripting languages
- ❑ **ASP.NET** – a server-side technology for creating dynamic web pages that lets you use any fully-fledged programming language supported by .NET
- ❑ **C#** – our chosen programming language for writing code in ASP.NET

Now it's time to get it all installed.

The Installation Process

Installation is going to be done in three steps. We're going to install the web server first, next we're going to install the prerequisites required for ASP.NET to work, and then lastly we're going to install either the **.NET Framework Redistributable** or the **.NET Framework SDK** (both of which contain ASP.NET), and are available for download from www.asp.net.

*SDK stands for **Software Development Kit**, and the only real difference between it and the **Redistributable** is the huge amounts of extra documentation and examples it supplies – 131MB compared with 21MB for its lightweight brother.*

Anybody who is familiar with ASP might be used to it being installed automatically with the web server, and thereby doing it all in one step. This is true – classic ASP is still installed with the web server. However, ASP.NET is currently only available as a separate download. This means you will have to download ASP.NET from Microsoft's web site or from CD (if you have one). However, before you can install ASP.NET, it is necessary to have a working web server.

If you have installed IIS 5.x already, or have installed either the Windows 2000 Server or Advanced Server operating system, then the good news is that you can skip this section, and go straight onto the section about installing the .NET Framework. However, for the rest of us, you will have to pay careful attention to the next section.

Installing the IIS 5.x Web Server

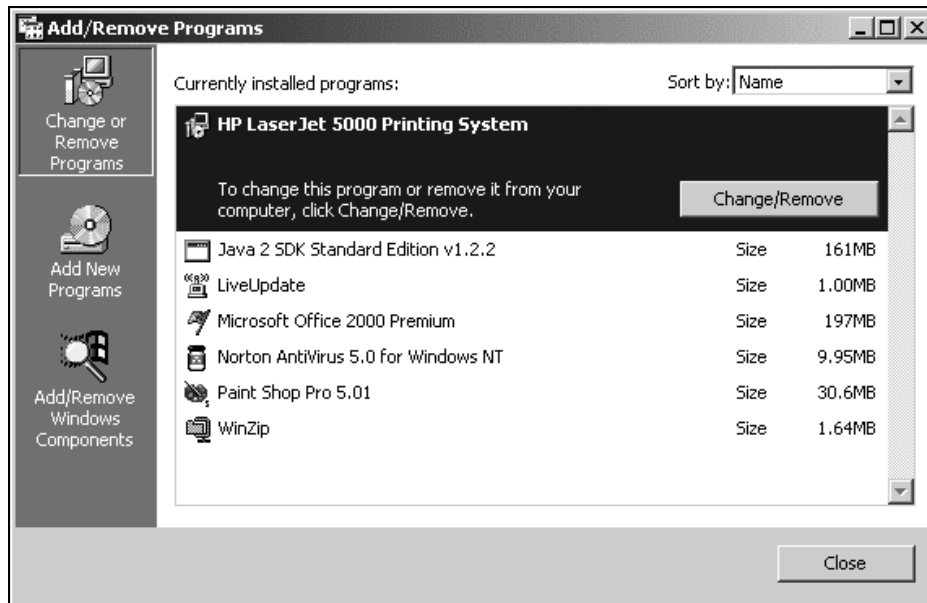
We'll look at the installation process for IIS on Windows 2000 Professional and Windows XP Professional together as they don't differ significantly. The main difference is that Windows 2000 installs IIS 5.0, while Windows XP installs IIS 5.1. The options for installation are exactly the same; the only thing that might differ is the look of the dialog boxes.

It's worth noting that you cannot install IIS on Windows XP Home Edition, and therefore you cannot run ASP.NET on it. It will only work on Windows XP Professional.

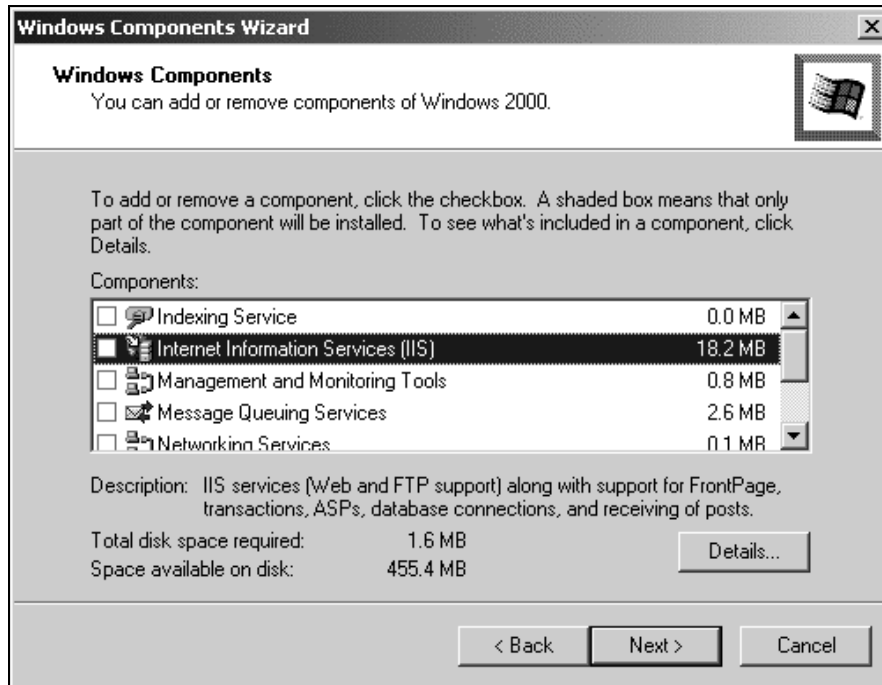
Before you install it though, it's worth noting that you might not have to do much in this initial stage, as it's possible you're already running IIS 5.x. We'll describe a process for checking whether this is the case as part of the installation process. You should also note that to install anything (not just ASP.NET, but literally anything) on Windows 2000/XP you need to be logged in as a user with administrative rights. If you're uncertain of how to do this, we suggest you consult your Windows documentation. Right let's get started!

Try It Out – Locating and/or Installing IIS 5.x on a Web Server Machine

1. Go to the control panel (Start | Settings | Control Panel) and select the Add/Remove Programs icon. The following dialog will appear, displaying a list of your currently installed programs:



2. Select the Add/Remove Windows Components icon on the left side of the dialog, to get to the screen that allows you to install new windows components:

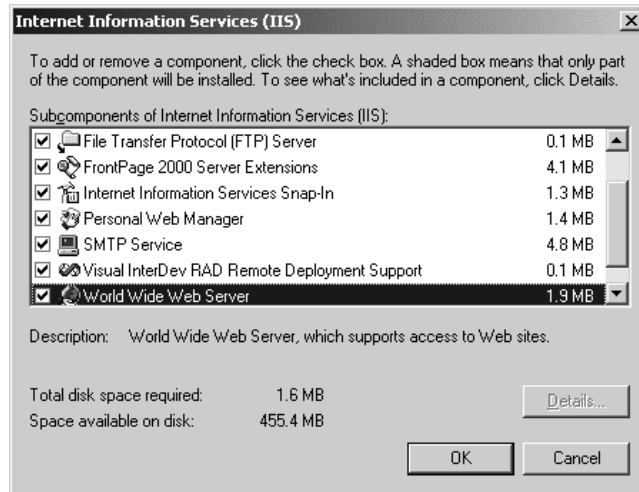


3. Locate the Internet Information Services (IIS) entry in the dialog, and note the checkbox that appears to its left. Unless you installed Windows 2000 via a custom install and specifically requested IIS, it's most likely that the checkbox will be unchecked (as shown above).
4. If the checkbox is *cleared*, then check the checkbox and click on Next to load Internet Information Services 5.x. You might be prompted to place your Windows 2000/XP installation disk into your CD-ROM drive. It will take a few minutes to complete. Then go to Step 5.

OR

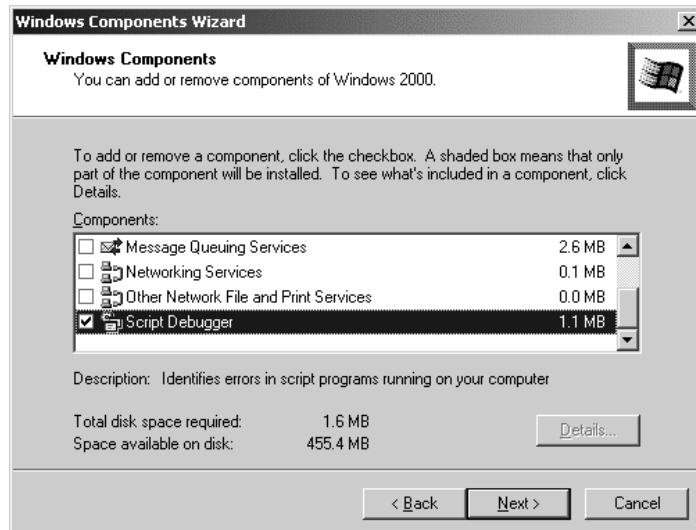
If the checkbox is *checked* then you won't need to install the IIS 5.x component – it's already present on your machine. Go to Step 6 instead.

- Click on the Details button – this will take you to the dialog shown here. There are a few options here, for the installation of various optional bits of functionality. For example, if the World Wide Web Server option is checked then our IIS installation will be able to serve and manage web pages and applications. If you're planning to use FrontPage 2000 or Visual InterDev to write your web page code, then you'll need to ensure that the FrontPage 2000 Server Extensions checkbox is checked. The Internet Information Services Snap-In is also very desirable, as you'll see later in the chapter, so ensure that this is checked too, the other options (although checked here) aren't necessary for this book.



For the purpose of this installation, make sure all the checkboxes in this dialog are checked. Then click on OK to return to the previous dialog.

- There's one other component that we'll need to install, for use later in this book – it's the Script Debugger. If you scroll to the foot of the Windows Components Wizard dialog that we showed above, you'll find a checkbox for Script Debugger. If it isn't already checked, check it now and click on Next to complete the installation. Otherwise, if both IIS 5.x and the script debugger are already present, you can click on Cancel to abort the process:



How It Works

IIS starts up automatically as soon as your installation is complete, and thereafter whenever you boot up Windows – so you don't need to run any further startup programs, or click on any short-cuts as you would to start up Word or Excel.

IIS installs most of its bits and pieces on your hard drive, under the `\WinNT\system32\inet_srv` directory. However, more interesting to us at the moment is the `\InetPub` directory that is also created at this time. This directory contains subdirectories that will provide the home for the web page files that we create.

If you expand the `InetPub` directory, you'll find that it contains several subdirectories:

- `\iissamples\homepage` contains some example classic ASP pages.
- `\iissamples\sdk` contains a set of subdirectories that hold classic ASP pages that demonstrate the various classic ASP objects and components.
- `\scripts` is an empty directory, where ASP.NET programs can be stored.
- `\webpub` is also empty. This is a 'special' virtual directory, used for publishing files via the Publish wizard. Note that this directory only exists if you are using Windows 2000 Professional Edition.
- `\wwwroot` is the top of the tree for your web site (or web sites). This should be your default web directory. It also contains a number of subdirectories, which contain various bits and pieces of IIS. This directory is generally used to contain subdirectories that hold the pages that make up our web site – although, in fact, there's no reason why you can't store your pages elsewhere.
- `\ftproot`, `\mailroot`, and `\nntproot` should form the top of the tree for any sites that use FTP, mail, or news services, if installed.
- In some versions of Windows, you will find an `\AdminScripts` folder, which contains various VBScript files for performing some common 'housekeeping' tasks on the web server, allowing you to stop and start services.

Working with IIS

Having installed IIS web server software onto our machine, we'll need some means of administering its contents and settings. In this section, we'll meet the user interface that is provided by IIS 5.x.

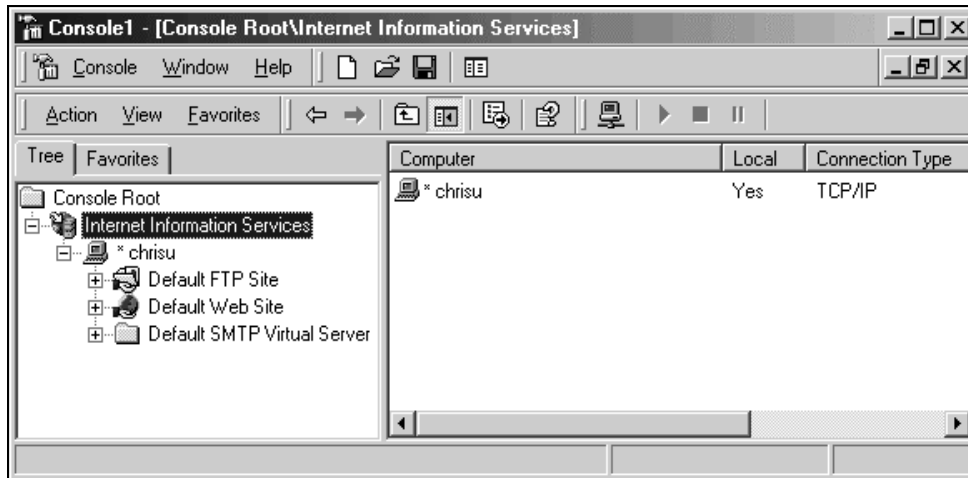
In fact, some versions of IIS 5.x provide two user interfaces, the MMC and the PWS interface. We're only going to look at one, as the other version is now obsolete. The version we will use is the **Microsoft Management Console (MMC)**, which that is a generic way of managing all sorts of services. Let's take a quick look at it now.

The Microsoft Management Console (MMC)

The beauty of the MMC is that it provides a central interface for administrating all sorts of services that are installed on your machine. We can use it to administer IIS – but in fact, when we use it to administer other services the interface looks roughly the same. The MMC is provided as part of the Windows 2000 operating system, and also comes with older Windows server operating systems.

Chapter 1

The MMC itself is just a shell – on its own, it doesn't do much at all. If we want to use it to administer a service, we have to add a **snap-in** for that service. The good news is that IIS 5.x has its own snap-in. Whenever you need to administer IIS, you can simply call up the Internet Services Manager MMC console by selecting **Start | Control Panel | Administrative Tools | Internet Services Manager**.



Having opened the IIS snap-in within the MMC, you can perform all of your web management tasks from this window. The properties of the web site are accessible via the **Default Web Site** node. We'll be using the MMC more a little later in the chapter.

Testing your Installation

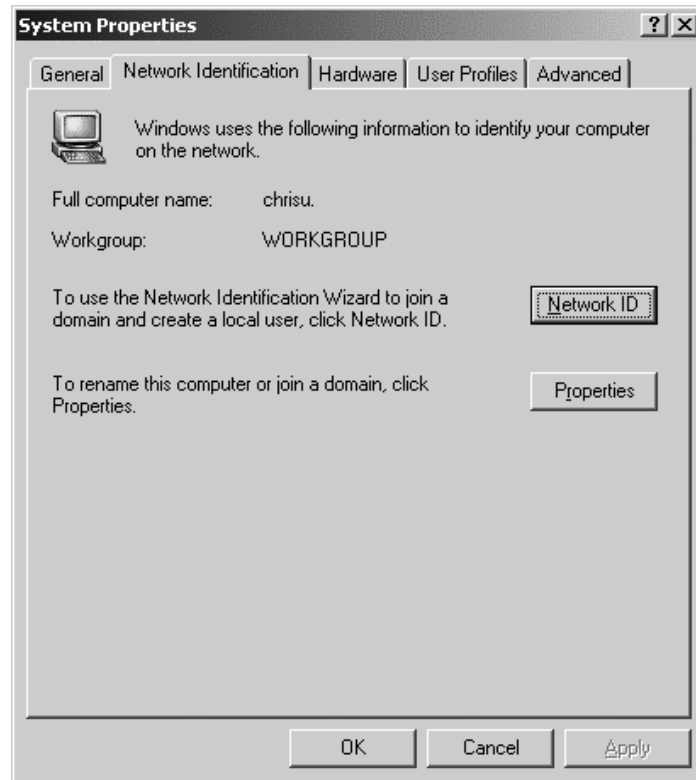
The next thing to do is test the web server to see if it is working correctly, and serving pages as it should. We've already noted that the web server should start as soon as IIS has been installed, and will restart every time you start your machine. In this section, we'll try that out.

In order to test the web server, we'll start up a browser and try to view some web pages that we know are already placed on the web server. In order to do that, we'll need to type a URL (Uniform Resource Locator) into the browser's **Address** box, as we often do when browsing on the Internet. The URL is an **http://...** web page address that indicates which web server to connect to, and the page we want to view.

What URL do we use in order to browse to our web server? If your web server and web browser are connected by a local area network, or if you're using a single machine for both web server and browser, then it should be enough to specify the name of the web server machine in the URL.

Identifying your Web Server's Name

By default, IIS will take the name of your web server from the name of the computer. You can change this in the machine's network settings. If you haven't set one, then Windows will generate one automatically – note that this automatic name won't be terribly friendly; probably something along the lines of "P77RTQ7881". To find the name of your own web server machine, select **Start | Settings | Network and Dial-up Connections** or **Start | Settings | Control Panel | System** (depending on which operating system you are using – if it isn't in one, try the other) and from the **Advanced** menu select **Network Identification**. The **Network Identification** tab will display your machine name under the description **Full computer name**:



My machine has the name `chrisu`, and (as you can see here and in the earlier screenshot of the MMC dialog) my web server has adopted the same name. On a computer within a domain you might see something different such as `WROX_UK/chrisu`, if the computer was in the `WROX_UK` domain. However, this doesn't alter operation for ASP.NET. Browsing to pages on this machine across a local area network (or, indeed, from the same machine), I can use a URL that begins `http://chrisu/...`

There are a couple of alternatives if you're using the same machine as both web server and browser. Try `http://127.0.0.1/...` – here, `127.0.0.1` is a default that causes requests to be sent to a web server on the local machine. Alternatively, try `http://localhost/...` – 'localhost' is an alias for the `127.0.0.1` address – you may need to check the LAN settings (in your browser's options) to ensure that local browsing is not through a proxy server (a separate machine that filters all incoming and outgoing web traffic employed at most workplaces, but not something that affects you if you are working from home).

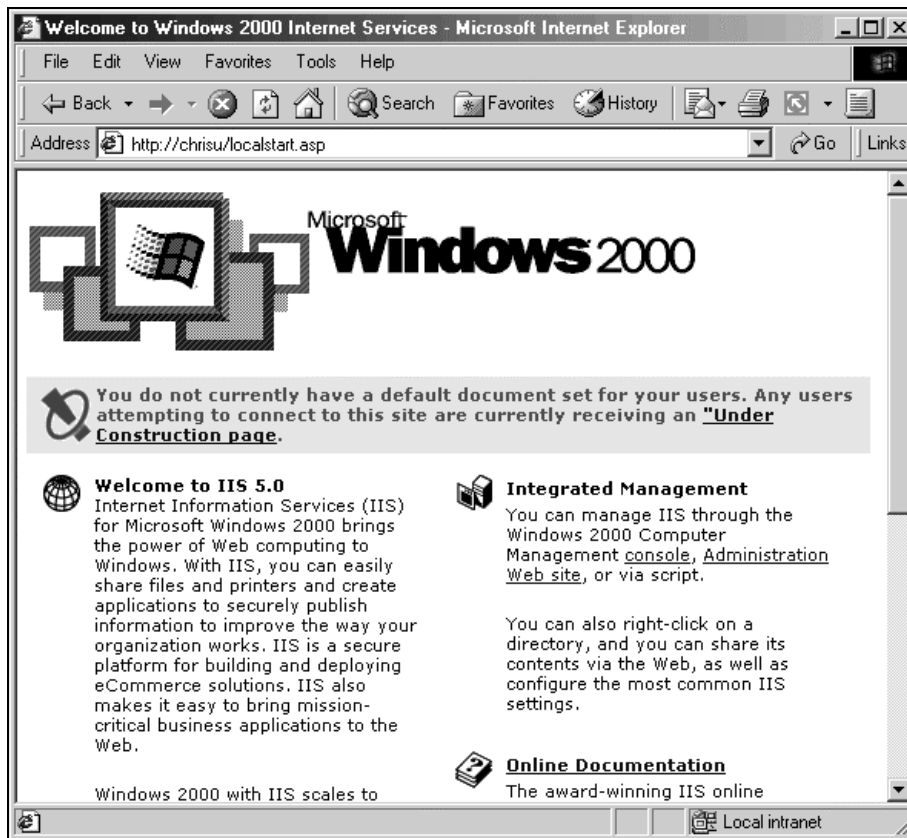
Throughout the book, in any examples that require you to specify a web server name, the server name will be shown as `localhost`, implicitly assuming that your web server and browser are being run on the same machine. If they reside on different machines, then you simply need to substitute the computer name of the appropriate web server machine.

Browsing to a Page on your Web Server

Now you know the name of your web server, and that web services are running; you can view some classic ASP pages hosted on your web server by browsing to them with your web browser. Let's test out this theory by viewing our default home page:

Try It Out – Testing the Web Service

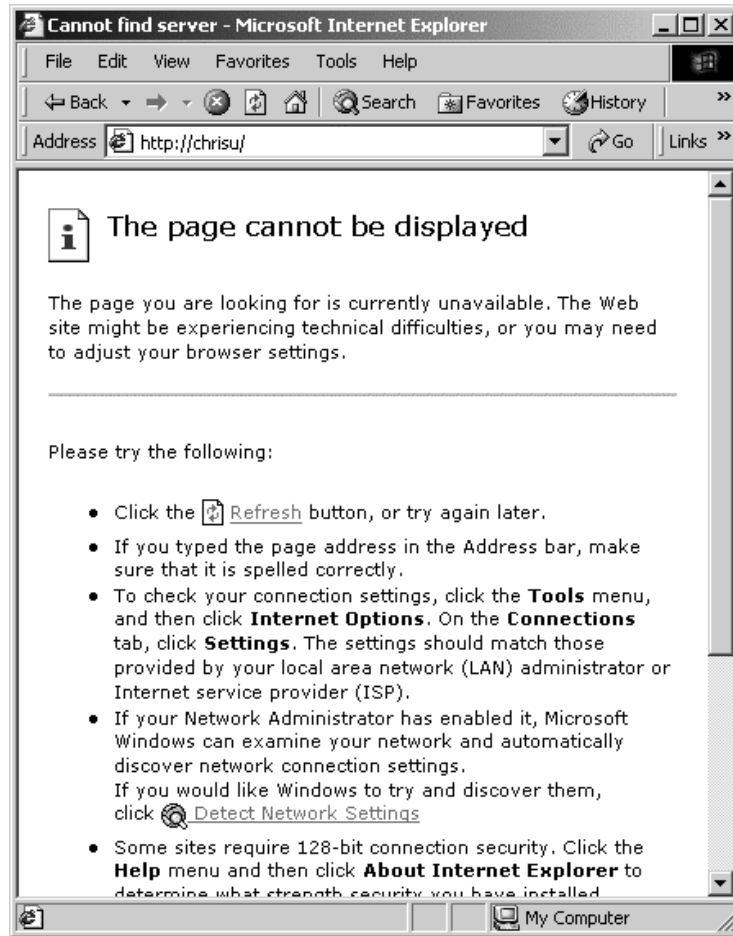
1. To verify that web services are working, start up your browser and type `http://my_server_name/localstart.asp` into the address box. (My server is named `chrisu`, so I typed in `http://chrisu/localstart.asp`.) Now press *Enter*; and (if all is well) you should get to see a page like this one:



Note that the default page we see here uses the .asp extension, denoting a Classic ASP page. Support for ASP3 is provided as part of the standard IIS5.x web server program.

What Do You Do If This Doesn't Work?

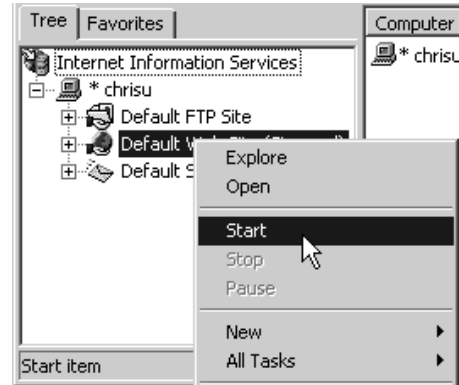
If you don't get this page, then take a look at the following steps as we try to resolve the problem. If it's not working correctly, then most likely you'll be greeted with this screen:



If you get this page then it can mean a lot of things. However, one of the most likely problems is that your web services under IIS are not switched on. To switch on web services, you'll first need to start the IIS admin snap-in that we described earlier in the chapter. (Select **Start | Run**, type **MMC** and hit **OK**, then select **Open** from the MMC's **Console** menu and locate the **iis.msc** file from the dialog. Alternatively, just use the shortcut that you created there.)

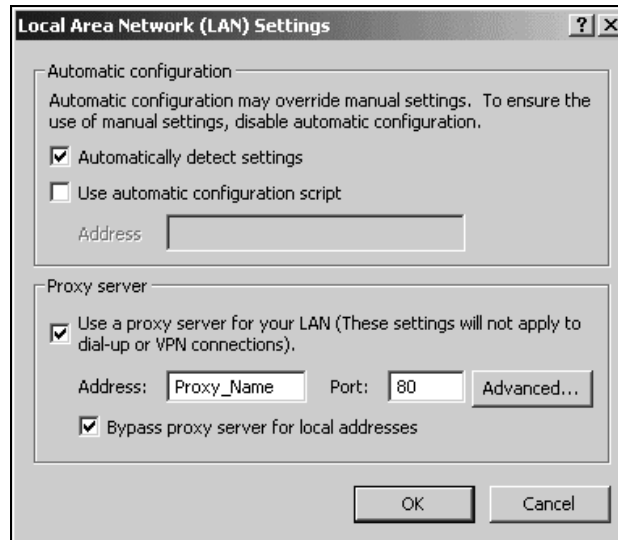
Chapter 1

Now, click on the + of the root node in the left pane of the snap-in, to reveal the Default sites. Then right-click on Default Web Site, and select Start:



If it's still not working then here are a few more suggestions about what could be wrong, which are based on particular aspects of your PC's setup. If you're running on a network and using a proxy server (a piece of software that manages connections from inside a firewall to the outside world – don't worry if you don't have one, they're mainly used by big businesses), there's a possibility that this can prevent your browser from accessing your web server. Most browsers will give you an opportunity to bypass the proxy server:

- ❑ If you're using Internet Explorer, you need to go to View | Internet Options (IE 4) or Tools | Internet Options (IE 5/IE 6) and select the Connections tab. In IE 5/IE 6 press the LAN Settings button and select Bypass the proxy server for local addresses. In IE 4, this section forms part of the Connections dialog:



- ❑ If you're using Netscape Navigator (either version 4.x or 6.x) and you are having problems then you need to turn off all proxies and make sure you are accessing the Internet directly. To do this, select **Edit | Preferences**. In the resulting dialog select **Advanced | Proxies** from the **Category** box on the left. Then on the right, select the **Direct Connection to Internet** option, and hit **OK**. Although you won't be browsing online to the Internet, it'll allow Netscape Navigator to recognize all variations of accessing local ASP.NET pages, such as `http://127.0.0.1`, `http://localhost`, and so on.

You may hit a problem if your machine name is similar to that of some web site out there on the Internet – for example, if your machine name is `jimmyd` but there also happens to be a public web site out there called `http://www.jimmyd.com`. When you type `http://jimmyd` into your browser's address box, expecting to view a page on your local web server, you unexpectedly get transported to `http://www.jimmyd.com` instead. If this is happening to you, then you need to make sure that you're not using a proxy server in your browser settings – again, this can be disabled using the **Internet Options | Connection** dialog or the **Edit | Preferences** dialog.

Lastly, if your web server is running on your home machine with a modem, and you get an error message informing you that your web page is offline, this could in fact be a misperception on the part of the web server. This can be corrected by changing the way that your browser looks for pages. To do this, select **View | Internet Options (IE 4)** or **Tools | Internet Options (IE 5/IE 6)**, choose the **Connections** tab and select **Never dial a connection**.

Of course, you might encounter problems that aren't answered above. In this case, the chances are that it's related to your own particular system setup. We can't possibly cover all the different possible configurations here, but if you can't track down the problem, you may find some help at one of the web sites and newsgroups listed later in this chapter.

Managing Directories on your Web Server

Before we install ASP.NET, we need to make one last pit stop in IIS. This is because when you come to run your ASP.NET pages, you need to understand where to place your pages, and how to make sure you have the permission to access them. As this is governed by IIS, now seems as good a time as any to investigate this.

These days, many browsers are sufficiently advanced that you can use them to locate and examine files and pages that exist on your computer's hard disk. So, for example, you can start up your browser, type in the physical location of a web page (or other file) such as `C:\My Documents\mywebpage.html`, and the browser will display it. However, this isn't real web publishing at all:

- ❑ First, web pages are transported using a protocol called HTTP – the HyperText Transfer Protocol. Note that the `http://` at the beginning of a URL indicates that the request is being sent by HTTP. Requesting `C:\My Documents\mywebpage.html` in your browser doesn't use HTTP, and this means that the file is not delivered and handled in the way a web page should be. No server processing is done in this case. We'll discuss this in greater detail when we tackle HTTP in Chapter 2.
- ❑ Second, consider the addressing situation. The string `C:\My Documents\mywebpage.html` tells us that the page exists in the `\My Documents` directory of the `C:` drive of the hard disk of **the machine on which the browser is running**. In a network situation, with two or more computers, this simply doesn't give enough information about the web server to locate the file.

Chapter 1

However, when a user browses (via HTTP) to a web page on some web server, the web server will need to work out where the file for that page is located on the server's hard disk. In fact, there's an important relationship between the information given in the URL, and the physical location (within the web server's file system) of the file that contains the source for the page.

Virtual Directories

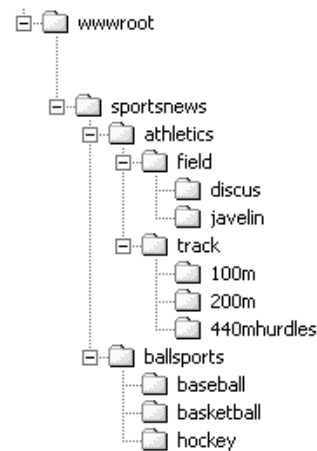
So how does the relationship between the information given in the URL, and the physical location of a file work? In fact, it can work by creating a second directory structure on the web server machine, which reflects the structure of your web site. It sounds like it could be complicated, but it doesn't have to be. In fact, in this book it's going to be very simple.

The first directory structure is what we see when we open Windows Explorer on the web server – these directories are known as **physical directories**. For example, the folder `C:\My Documents` is a physical directory.

The second directory structure is the one that reflects the structure of the web site. This consists of a hierarchy of **virtual directories**. We use the web server to create virtual directories, and to set the relationship between the virtual directories and the real (physical) directories.

When you try to visualize a virtual directory, it's probably best not to think of it as a directory at all. Instead, just think of it as a nickname or alias for a physical directory that exists on the web server machine. The idea is that, when a user browses to a web page that is contained in a physical directory on the server, they don't use the name of the **physical** directory to get there, instead, they use the physical directory's nickname.

To see how this might be useful, consider a web site that publishes news about many different sporting events. In order to organize the web files carefully, the Webmaster has built a physical directory structure on the hard disk, which looks like this:



Now, suppose you visit this web site to get the latest news on the Javelin event in the Olympics. If the URL for this web page were based on the physical directory structure, then the URL for this page would be something like this:

<http://www.oursportsite.com/sportsnews/athletics/field/javelin/default.asp>

That's OK for the Webmaster, who understands his directory structure, but it's a fairly unmemorable web address! So, to make it easier for the *user*, the Webmaster can assign a *virtual* directory name or *alias* to this directory – it acts just like a nickname for the directory. Here, let's suppose we've assigned the virtual name `javelinnews` to the `c:\inetpub\...\javelin\` directory. Now, the URL for the latest Javelin news is:

`http://www.oursportsite.com/javelinnews/default.asp`

By creating virtual directory names for all the directories (such as `baseballnews`, `100mnews`, `200mnews`, and so on) it's easy for the user to type in the URL and go directly to the page they want:

`http://www.oursportsite.com/baseballnews/default.asp`

`http://www.oursportsite.com/100mnews/default.asp`

`http://www.oursportsite.com/200mnews/default.asp`

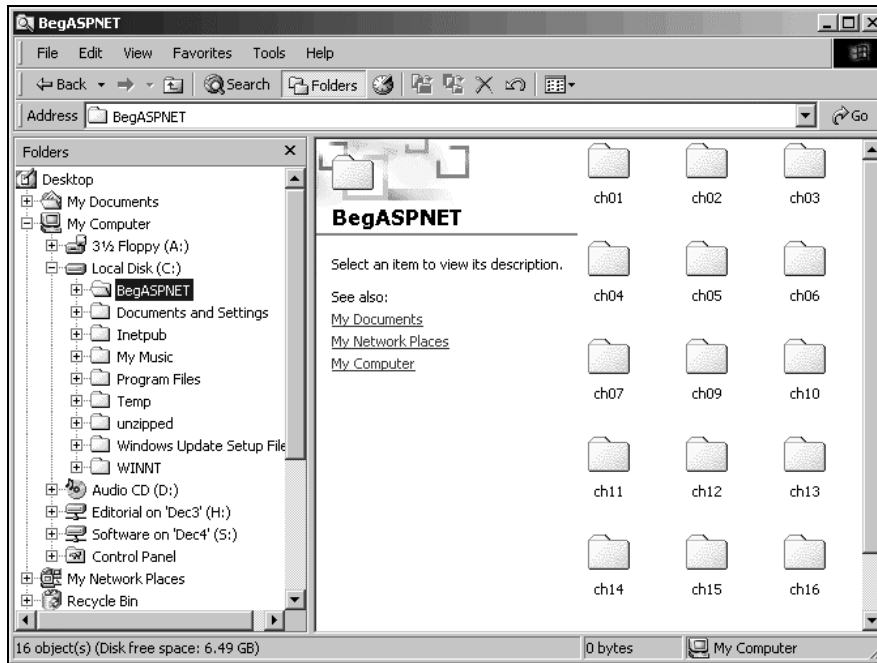
Not only does this save the user from long, unwieldy URLs – it also serves as a good security measure, because it hides the physical directory structure from all the web site visitors. This is good practice, otherwise hackers might be able to work out and access our files if they knew what the directory structure looked like. Moreover, it allows the Webmaster's web site structure to remain independent of the directory structure on their hard drive – so they can move files the between different physical folders, drives, or even servers, without having to change the structure of his web pages. There is a performance overhead to think about as well, as IIS has to expend effort translating out the physical path. It can be a pretty costly performance-wise to have too many virtual directories.

Let's have a crack at setting up our own virtual directories and permissions (please note that these permissions are set automatically if you use the FrontPage editor to create a new site – so don't use FrontPage to set up this site for you unless you know what you're doing).

Try It Out – Creating a Virtual Directory and Setting up Permissions

Let's take a quick look now at how you can create your own virtual directory. We'll use this directory to store the examples that we'll be creating in this book. We don't want to over complicate this example by creating lots of directories, so we'll demonstrate by creating a single physical directory on the web server's hard disk, and using the IIS admin tool to create a virtual directory and make the relationship between the two:

1. Start Windows Explorer and create a new physical directory named `BegASPNET`, in the root directory of your hard drive. For example, `C:\BegASPNET\`:



2. Next, start up the IIS admin tool (using the MMC, as we described earlier). Right-click on Default Web Site, and from the menu that appears select New | Virtual Directory. This starts the Virtual Directory Creation Wizard, which handles the creation of virtual directories for you and the setting up of permissions as well. You'll see the splash screen first, which looks like this. Click on Next:



3. Type BegASPNET in the Alias text box; then click Next:



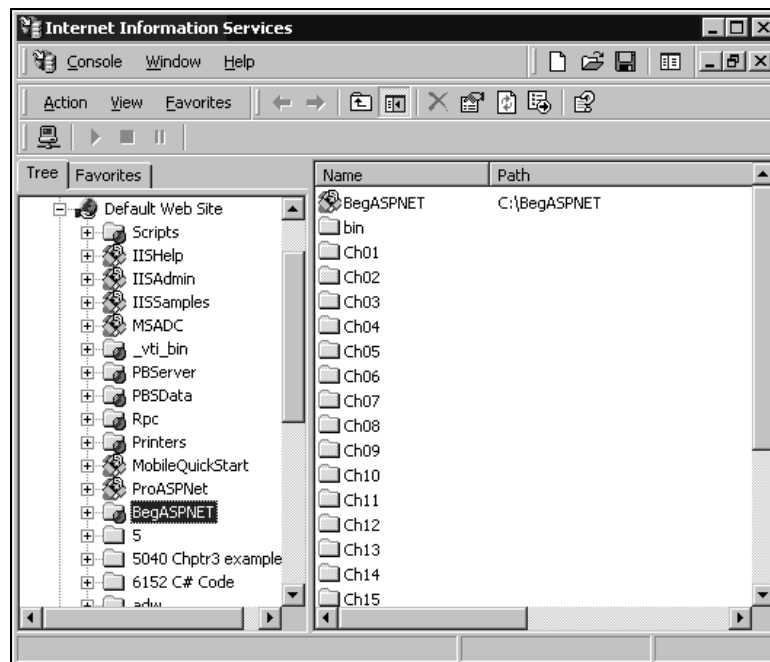
4. Click on the Browse... button and select the directory \BegASPNET that you created in Step 1. Then click Next:



5. Make sure that the **Read** and **Run scripts** checkboxes are checked, and that the **Execute** checkbox is empty. Click on **Next**, and in the subsequent page click on **Finish**:



6. The BegASPNET virtual directory will appear on the tree in the IIS admin window:



How It Works

You've just created a physical directory called `BegASPNET`; this directory will be used throughout the book to store our code examples. The download files from `wrox.com` are also designed to follow this structure. Within this directory we recommend that you create a subdirectory for each of the chapters in order to keep things tidy (this needn't be a virtual directory – just a physical one.)

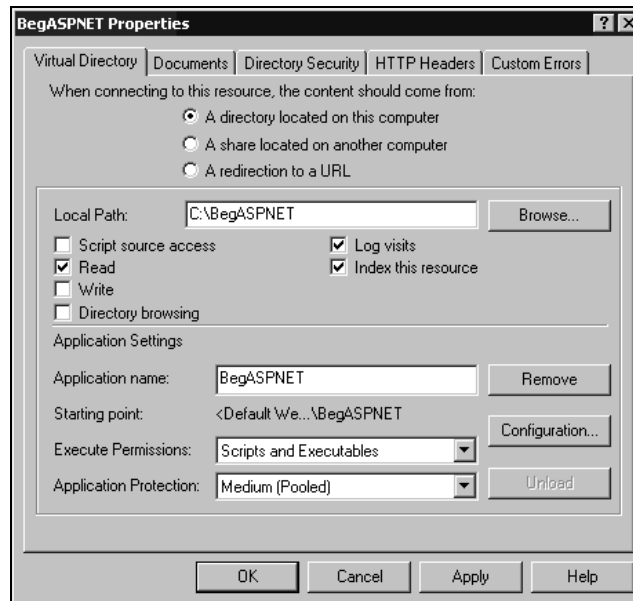
You've also created a virtual directory called `BegASPNET`, which you created as an alias for the physical `BegASPNET` directory. If when we create Chapter 1 examples you place the ASP.NET files in the physical `C:\BegASPNET\Ch01`, directory, you can use the browser to access pages stored in this folder. You'll need to use the URL `http://my_server_name/BegASPNET/Ch01/...`

You should also note that the URL uses the alias `/BegASPNET` – IIS knows that this stands for the directory path `C:\BegASPNET`. When executing ASP.NET pages, you can reduce the amount of typing you need to do in the URL, by using virtual directory names in your URL in place of the physical directory names.

We also set the permissions `Read` and `Run` – these must be set or the IIS security features will prevent you from running any ASP.NET pages. The `Execute` checkbox is left empty as allowing others to run applications on your own machine is a sure way of getting viruses or getting hacked. We'll take a closer look at permissions now, as they are so important. If you don't assign them correctly you may find that you're unable to run any ASP.NET pages at all – or worse still, that anybody at all can access your machine, and alter (or even delete) your files via the Web.

Permissions

As we've just seen, we can assign permissions to a new directory as we create it, by using the options offered in the Virtual Directory Wizard. Alternatively, we can set permissions at any time, from the IIS admin tool in the MMC. To do this, right-click on the `BegASPNET` virtual directory in the IIS admin tool, and select `Properties`. You'll get the following dialog:



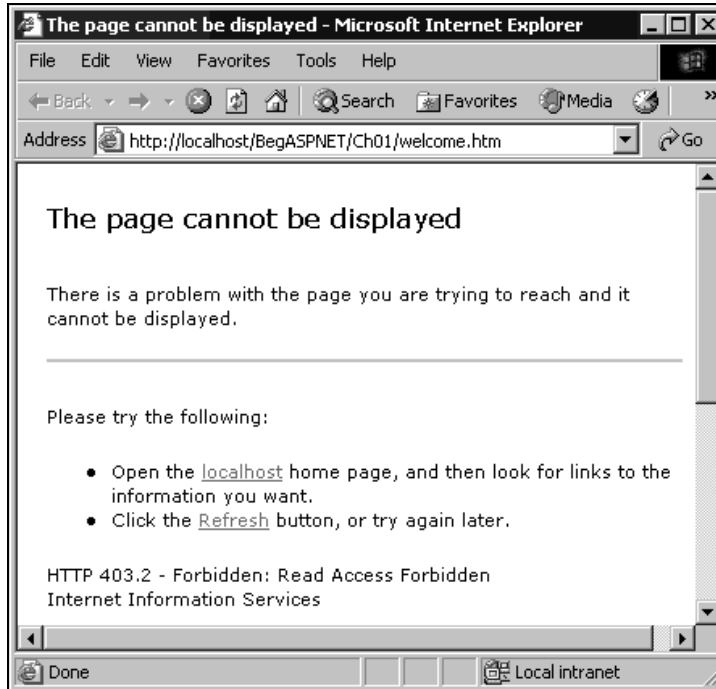
It's quite a complicated dialog, and it contains a lot of options – not all of which we wish to go into now.

Access Permissions

The four checkboxes on the left are of interest to us, as they govern the types of access for the given directory and dictate the permissions allowed on the files contained within that directory. Let's have a look at what each of these options means:

- ❑ **Script source access** – This permission enables users to access the source code of an ASP.NET page. It's only possible to grant this permission if the **Read** or **Write** permission has already been assigned. But we generally don't want our users to be able to view our ASP.NET source code, so we would usually leave this checkbox unchecked for any directory that contains ASP.NET pages. By default, all directories created during setup have **Script Source Access** permission disabled. You should leave this as is.
- ❑ **Read** – This permission enables browsers to read or download files stored in a home directory or a virtual directory. If the browser requests a file from a directory that *doesn't* have the **Read** permission enabled, then the web server will simply return an error message. Note that when the folder has **Read** permission turned off, HTML files within the folder cannot be read, but ASP.NET code within the folder can still be run. Generally, directories containing information that you want to publish (such as HTML files, for example) should have the **Read** permission enabled, as we did in our *Try It Out*.
- ❑ **Write** – If the write permission on a virtual directory is enabled, then users will be able to create or modify files within the directory, and change the properties of these files. For reasons of security, this is not normally turned on, and we don't recommend you alter it.
- ❑ **Directory Browsing** – If you want to allow people to view the contents of the directory (that is, to see a list of all the files that are contained in that directory), then you can allow this by checking the **Directory Browsing** option.

If someone tries to browse the contents of a directory that has **Directory Browsing** enabled but **Read** disabled, then they may receive the following message:

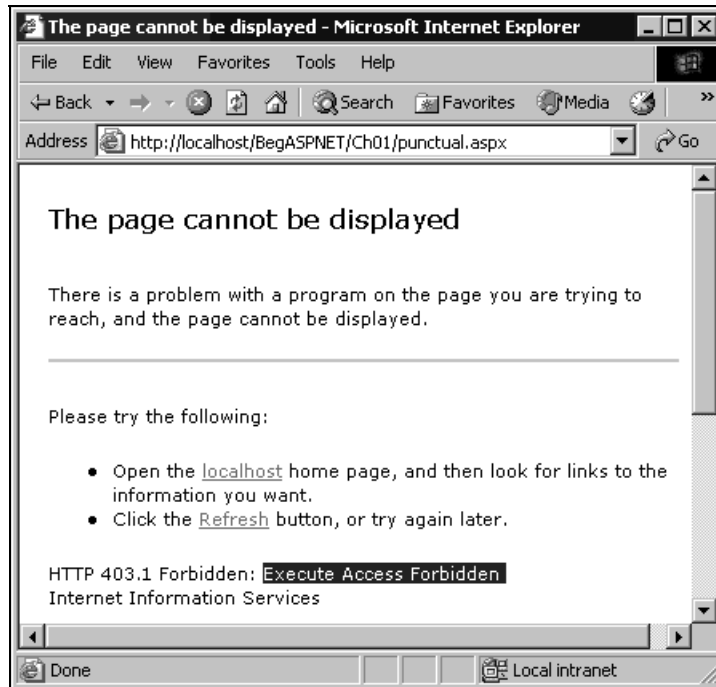


For security reasons, we'd recommend disabling this option unless your users specifically need it – such as when transferring files using FTP (file transfer protocol), from your web site . If you don't know what FTP is then, we strongly recommend that you disable it, as you wont need it!

Execute Permissions

There's a drop-down listbox near the foot of the Properties dialog, labeled Execute permissions. This specifies what level of program execution is permitted on pages contained in this directory. There are three possible values here – None, Scripts only, or Scripts and Executables:

- ❑ Setting Execute permissions to None means that users can only access static files, such as image files and HTML files. Any script-based files or other executables contained in this directory are inaccessible to users. If you tried to run an ASP.NET page, from a folder with the permission set to None, you would receive the following – note the Execute Access Permission forbidden message in the page:



- ❑ Setting **Execute** permissions to **Scripts Only** means that users can also access any script-based pages, such as ASP.NET pages. So if the user requests an ASP.NET page that's contained in this directory, the web server will allow the ASP.NET code to be executed, and the resulting HTML to be sent to the browser.
- ❑ Setting **Execute** permissions to **Scripts and Executables** means that users can execute any type of file type that's contained in the directory. It's generally a good idea to avoid using this setting, in order to prohibit users from executing potentially damaging applications on your web server.

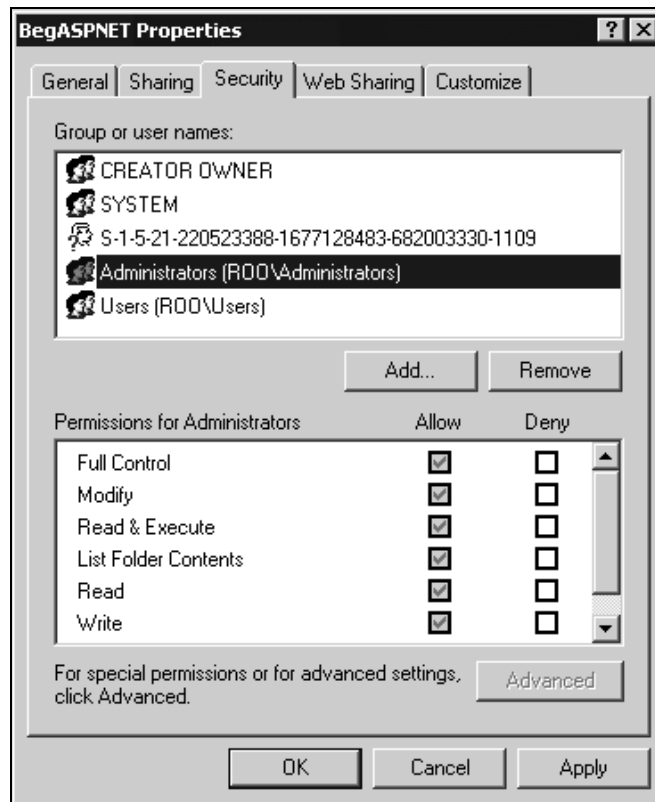
For any directory containing ASP.NET files that you're publishing, the appropriate setting for the **Execute** permissions is **Scripts Only**. There is one last thing about directories that needs pointing out though.

Configuring Directory Security

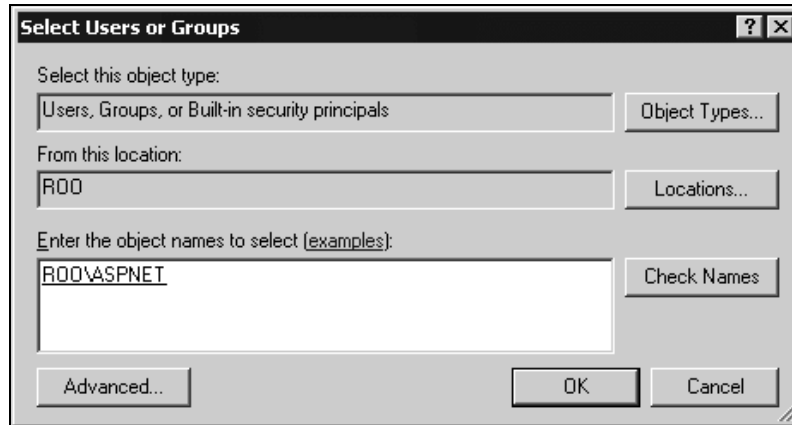
For the release version of ASP.NET, all ASPX pages run under a special user account with the name of **ASPNET**. For security reasons, by default this account has restricted permissions, and ordinarily this isn't a problem. However, the database samples in this chapter use **Access**. When updating data in an **Access** database, a separate file is created (with a suffix of **.ldb**), which holds the locking information. These are the details that store who is updating records, and the locking file is created and removed on demand.

The security problem we encounter is that while running pages we are running under the ASPNET account, and this account doesn't have write permissions in the samples directory. Consequently any ASP.NET pages that update the `Northwind.mdb` database will fail. Setting the write permission is simple – just follow these steps:

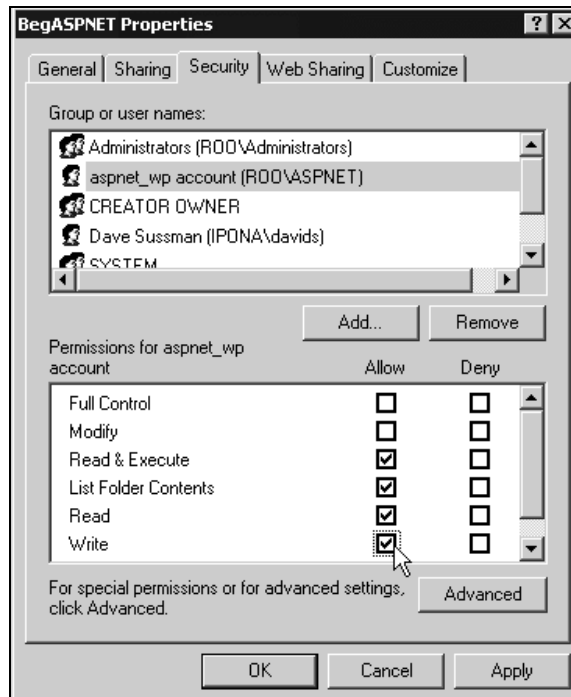
1. In Windows Explorer, select the `BegASPNET` directory, where the samples are located.
2. Using the right mouse button, select the `Properties` menu option, and from the `Properties` dialog that appears, select the `Security` tab:



3. Click the Add button to display the Select Users or Groups dialog. In the blank space enter ASPNET and click the Check Names button. This checks the name you've entered and adds the machine name to it:



4. Click the OK button to return to the Properties dialog, and you'll see that the ASPNET user is now shown in the list of users. In the Permissions area, at the bottom of this screen, select the Write permission and tick it. This gives the ASPNET user write permission to the BegASPNet directory tree:



5. Click the OK button to save the changes, and to close the dialog.

This security issue is only a problem if you need write access to a directory, as both Access and our XML samples do. Most production web sites wouldn't use Access as their database store, since Access isn't designed for a high number of users. In these cases it's more likely that SQL Server will be used. Luckily, the only changes you'd have to make to the code samples shown in this chapter is to the connection string, which contains the details of how to connect to the database. The .NET SDK documentation has examples of connection strings for SQL Server.

Now you've started to familiarize yourself with IIS, you're ready to prepare your machine for the installation of ASP.NET itself.

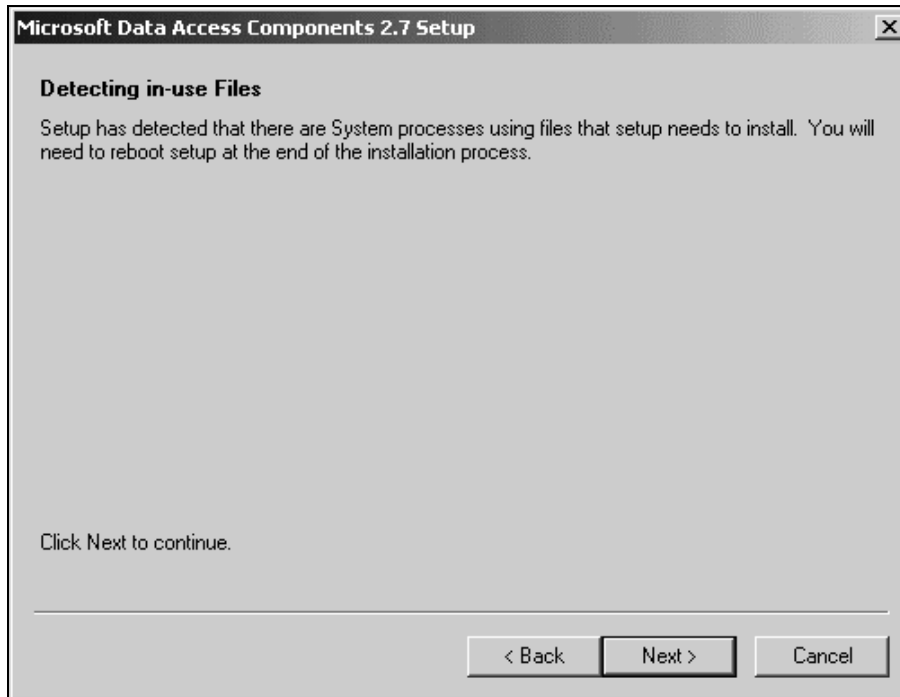
Prerequisites for Installing ASP.NET

Before you can install ASP.NET or the .NET Framework you will need to install the Microsoft Data Access Components (MDAC) version 2.7 or later. This is a set of components that will enable you to use ASP.NET to communicate with databases and display the contents of your database on a web page. Without these components installed you won't be able to run any of the database examples in this book. This will affect examples as early as Chapter 2, so please don't skip this stage! Although you might already have an earlier version of MDAC installed (such as 2.5 if you're using Windows 2000), unless you have specifically upgraded, in all likelihood you won't have the most up-to-date version and will still need to upgrade.

The Microsoft Data Access Components is a small download (roughly 5 or 6 MB) available for free from Microsoft's site at www.microsoft.com/data. The installation for MDAC 2.7 is pretty straightforward, and it also comes as part of the Windows Component Update of the .NET Framework, but we'll run through it quickly just to make sure that everything is clear.

Try It Out – Installing MDAC 2.7

1. MDAC 2.7 comes as a single file `MDAC_typ.exe` that you will need to run. If you run this EXE file, then it will begin the installation process.
2. After agreeing to the terms of the license, there's a good chance that you will be asked to reboot your system, it will tell you this in advance:



3. Then the installation process will continue without requiring further intervention, although you might have to wait for a system reboot, if one was specified earlier.

You're now ready to install ASP.NET.

Installing ASP.NET and the .NET Framework SDK

We're almost ready to install ASP.NET, but there are two important points to be made beforehand.

First, there are two different types of installation available from Microsoft's www.asp.net site, the .NET Framework SDK and .NET Framework Redistributable. Both downloads contain ASP.NET, C#, and the .NET Framework.

The **.NET Framework Redistributable** download is a smaller, streamlined download that only contains the bare bones needed for you to run ASP.NET and the .NET Framework. None of the extra documentation or samples will be included. The size differential between the two is pretty big (the Redistributable is 21MB while the .NET Framework SDK is a staggering 131MB), so unless you have the .NET Framework SDK on CD (which you can order from the Microsoft site) or broadband high-speed Internet access, you'll probably want to download the Redistributable version.

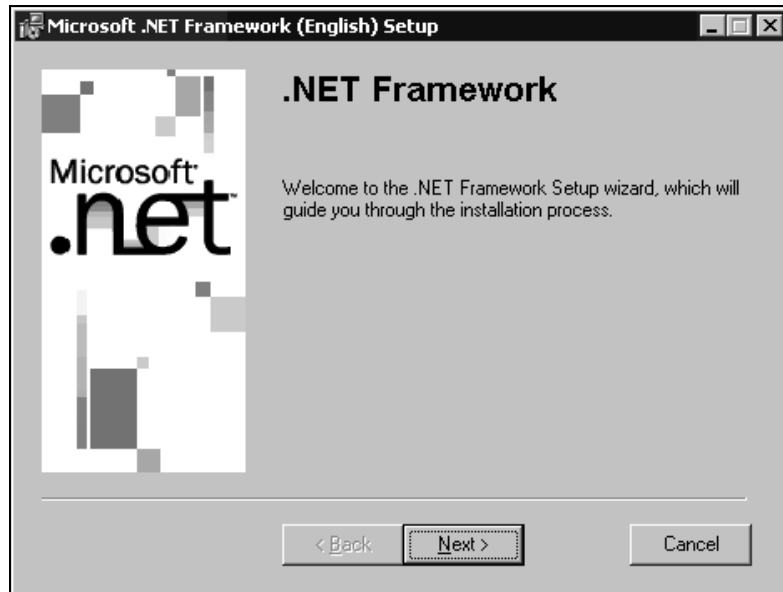
Don't worry, this won't affect your ability to run the examples in this book – everything's been written so that it will run on the .NET Framework Redistributable version of ASP.NET. While you won't have direct access to the help files, all support materials are available online at Microsoft's www.asp.net site.

Also, don't worry that you might end up replacing an existing Classic ASP installation, since ASP.NET will be installed alongside ASP and they will both continue to work with no action from us.

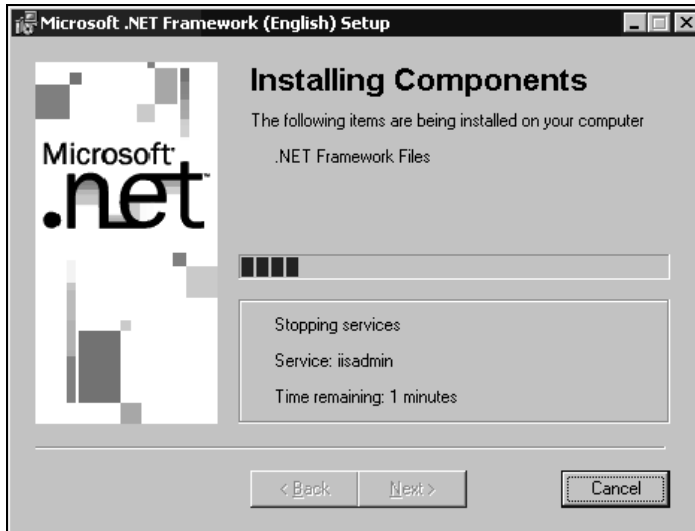
We'll now walk you through a typical installation of both .NET Framework Redistributable and the .NET Framework SDK. The installation process is the same for Windows 2000 and Windows XP, so once again we're only going to detail the installation process on the former. Although the wizard looks a bit different on XP, it asks for exactly the same things.

Try It Out – Installing the .NET Framework Redistributable

1. After downloading, click on the installation file (currently called `dotnetfx.exe`), you will be asked to confirm your intent, and then, after a short interval, you are propelled into the setup wizard:



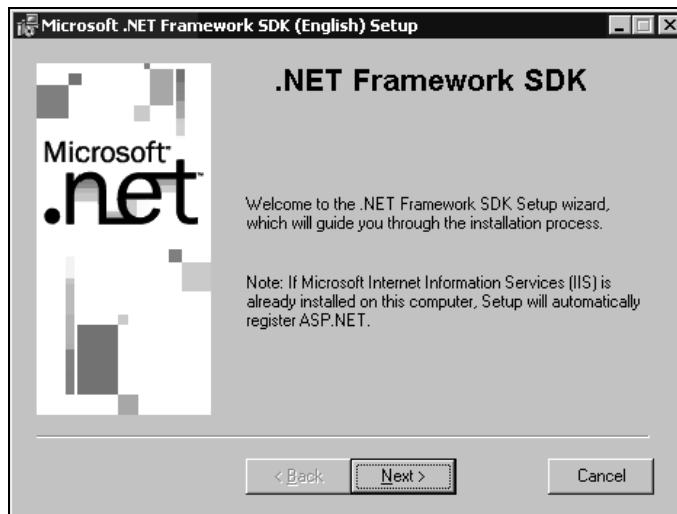
2. Click on Next and accept the License agreement to continue. ASP.NET will now install without further intervention:



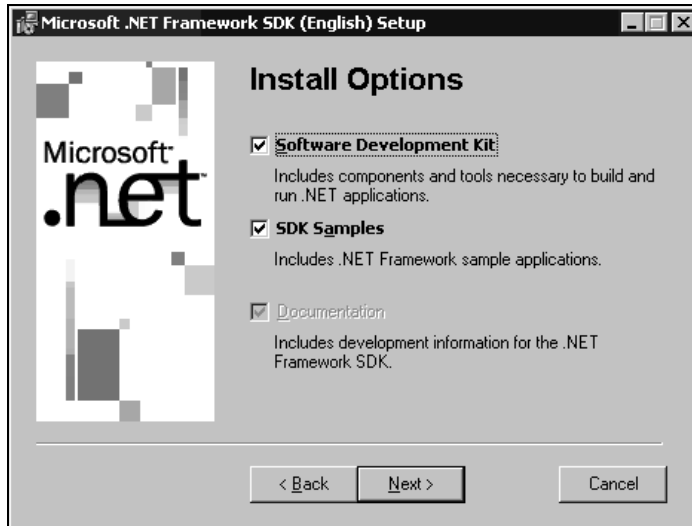
3. You will be notified when installation has finished, and unlike with MDAC 2.7, you probably won't have to reboot. You can now jump to the testing section, later in this chapter, to check everything is working.

Try It Out – Installing the .NET Framework SDK

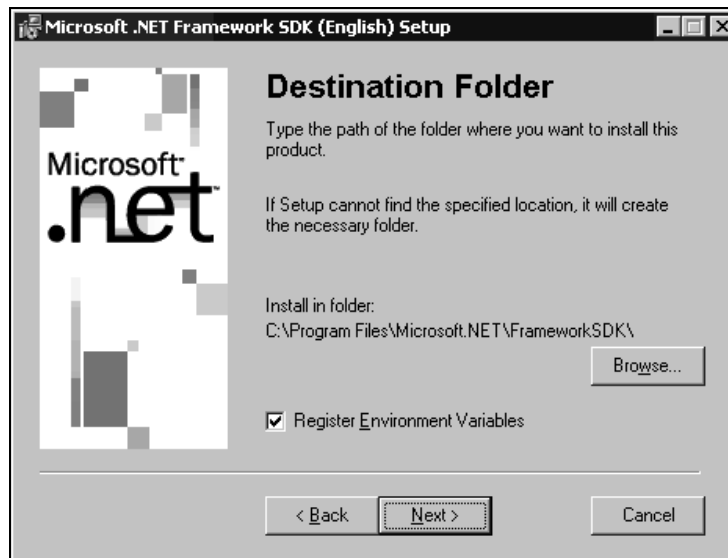
1. After downloading, click on setup.exe and confirm that you do want to install NET Framework SDK package. After an interval of a few minutes, you are propelled into the setup wizard:



2. Click on Next and accept the License agreement to continue. The next dialog after the license agreement will ask you which different pieces of the SDK you need to install. You should check all of them, although if you're short of hard drive space, you could choose to omit the SDK_Samples or Documentation. The Software Development Kit is essential:



3. After clicking on Next you get to a dialog that specifies the destination folder for the different .NET Framework SDK samples and bits and pieces. You can choose to install these wherever you want. More importantly there is a checkbox at the foot of the dialog, which asks you to Register Environment Variables. This checkbox should be checked, as we will use the environment variables in later chapters:



4. Click on Next and the .NET Framework SDK will install without further ado. It shouldn't require a reboot.

Troubleshooting Hints and Tips

The installation process is very straightforward, and will work on the majority of machines. However, sometimes the particular configuration of your machine will prevent it from installing. Unfortunately we can't cover all of the possible eventualities, but if it doesn't work on your machine, you should check that you have enough hard disk space, as this is the most common cause of problems. Also try to ensure that the installation process isn't curtailed half way, as no installer is completely foolproof at removing all the different bits and pieces of the aborted install, and this can cause problems when you try to reinstall, and leave you needing to reformat your hard drive to get it to work correctly. Other than that, check the list of newsgroups and resources later in this chapter.

ASP.NET Test Example

OK, we've now reached the crux of the chapter, checking to see if everything is working correctly. Do you remember the punctual web server code that we talked about earlier in the chapter – in which we wanted to write a web page that displays the current time? We'll return to that example now. As you'll see it's quite a simple bit of code, but it should be more than enough to check that ASP.NET is working OK.

Try It Out – Your First ASP.NET Web Page

1. Open up a text editor and type in the following code exactly as you see it, including case, as C# is case-sensitive and will reject `Void` instead of `void`, for instance:

```
<script language="c#" runat="server">
void Page_Load()
{
    time.Text=DateTime.Now.Hour.ToString() + ":" +
        DateTime.Now.Minute.ToString() + ":" +
        DateTime.Now.Second.ToString();
}
</script>

<html>
<head><title>The Punctual Web Server</title></head>
<body>
    <h1>Welcome</h1>
    In WebServerLand the time is currently:
    <asp:label id="time" runat="server" />
</body>
</html>
```

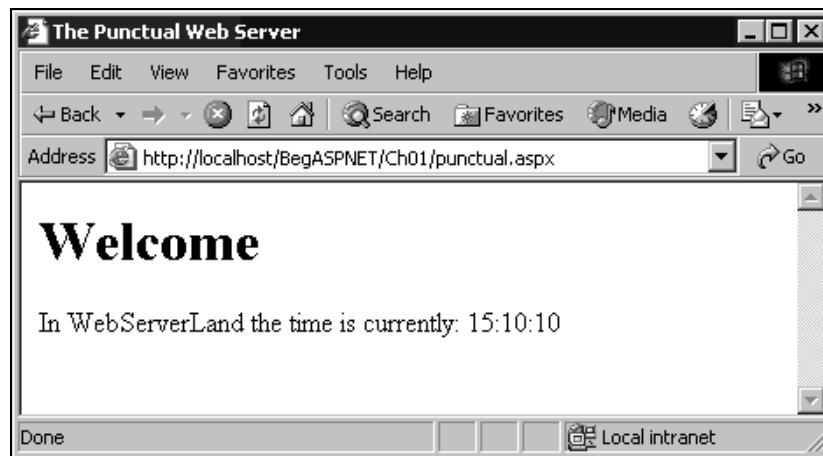
We strongly suggest (and will assume throughout) that you use Notepad to code all the examples in this book, since it will always do precisely what you ask it to and no more. It's therefore a lot easier to track down any problems you're having, and is a great deal easier than troubleshooting problems caused by FrontPage or similar web page editors.

2. Save this page as `punctual.aspx`. Make sure that you save it in the physical folder you created earlier, `C:\BegASPNET\Ch01\`.

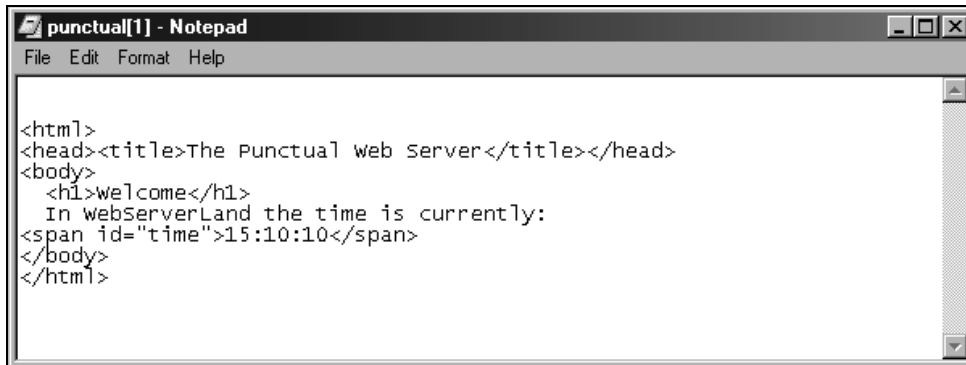
When you save the file, you should double-check that your new file has the correct suffix. It should be `.aspx`, since this is how you tell the web server that the page contains ASP.NET code. Be aware that Notepad (and many other text editors) consider `.txt` to be the default. So in the **Save** or **Save As** dialog, make sure that you change the **Save As** type to read **All Files**, or **All Files (*.*)**, or enclose the path and filename in quotes.

3. Now start up your browser and type in the following:

`http://localhost/BegASPNET/Ch01/punctual.aspx`



4. Click on the refresh button of the browser and the displayed time will change. In effect the browser is showing a new and different instance of the same page.
5. Now on your browser select **View Source** or similar (depending on which browser you're using) from the browser menu to see the HTML source that was sent from the web server to the browser. The result is shown below. You can see that there is no ASP.NET code to be seen, and nothing before the first `<html>` tag – the ASP.NET code has been processed by the web server and used to generate pure HTML, which is hard-coded into the HTML source that's sent to the browser:



```
punctual[1] - Notepad
File Edit Format Help

<html>
<head><title>The Punctual web Server</title></head>
<body>
  <h1>welcome</h1>
  In webserverLand the time is currently:
  <span id="time">15:10:10</span>
</body>
</html>
```

Here, you can see the HTML that was sent to the browser when I refreshed the page at 15:10:10.

6. As we mentioned before, you can expect this to work in any browser – because ASP.NET is processed on the web server. If you have another browser available, give it a go!

How It Works

Easy wasn't it? If you didn't get it to work first time, then don't rush off to e-mail technical support just yet – have a little look at the next section, *ASP.NET Troubleshooting*, first. Now let's take a look at the ASP.NET code that makes this application tick.

There is only one block of ASP.NET code (ignoring the server control – we'll look at using server controls in Chapter 3) in the whole program. It's enclosed by the `<script>` and `</script>` tags and although it does span three lines it is actually only one line of code spread over three lines, because it won't fit within the margins of the book on one line:

```
<script language="c#" runat="server">
void Page_Load()
{
    time.Text=DateTime.Now.Hour.ToString() + ":" +
        DateTime.Now.Minute.ToString() + ":" +
        DateTime.Now.Second.ToString();
}
</script>
```

The script delimiters specify which code is to be run by ASP.NET. We'll look at these in detail in the next chapter. If we ignore the `<script>` tags for the time being, and just think of them as ASP.NET code delimiters, then we're left with just three lines. If we further ignore the `void Page_Load()` and surrounding curly braces `{ }`, which are standard to many ASP.NET programs, and which we'll be discussing in Chapter 3, we're left with this:

```
time.Text=DateTime.Now.Hour.ToString() + ":" +
    DateTime.Now.Minute.ToString() + ":" +
    DateTime.Now.Second.ToString();
```

This code tells the web server to go off and run the C# `DateTime.Now()` function **on the web server**. The C# `DateTime.Now()` function returns the current time **at the web server**. We can divide this up to return values for the hours, minutes, and seconds, using the `DateTime.Now.Time` format. We also add a `ToString`, because as it stands, we can't return a date data type, it has to be a string data type. We will look at data types in Chapter 4. Anyway we don't really want to dig into the code here, all we need to know is that the code tells the web server to go and get the system time and return it as three values, each converted to a string and separated by a colon. The resulting string looked like this:

15:39:15

The formatted string is returned as part of the `<ASP: label>` control, further down the page. We'll be looking at this control in Chapter 3.

If the web server and browser are on different machines, then the time returned by the web server might not be the same as the time kept by the machine you're using to browse. For example, if this page is hosted on a machine in Los Angeles, then you can expect the page to show the local time in Los Angeles, even if you're browsing to the page from a machine in Cairo.

This example isn't wildly interactive or dynamic, but it illustrates the way in which we can ask the web server to go off and do something for us, and return the answer **within the context** of an HTML page. Of course, by using this technique with things like HTML forms and other tools, we'll be able to build a more informative, interactive interface with the user.

ASP.NET Troubleshooting

If you had difficulty with the example above, then perhaps you fell into one of the simple traps that commonly snare new ASP.NET programmers, and that can easily be rectified. In this section we'll look at a few common errors and reasons why your script might not run. If you did have problems, maybe this section will help you to identify them.

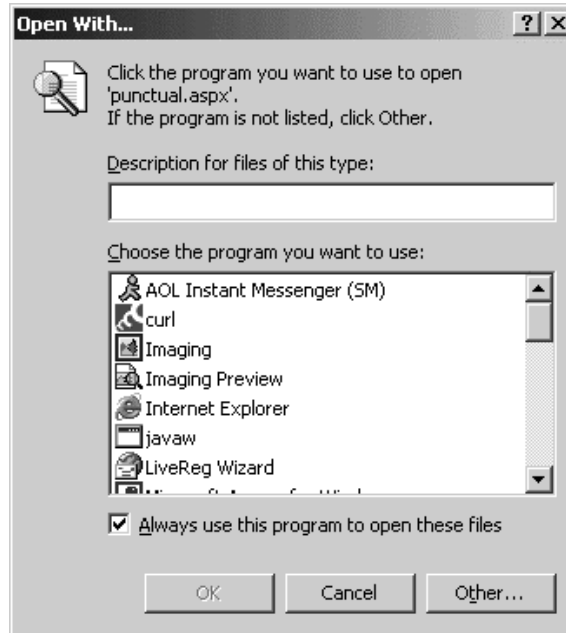
Program Not Found, the Result of the ASP.NET Isn't Being Displayed, or The Browser Tries to Download the File

You'll have this problem if you try to view the page as a local file on your hard drive, like this:

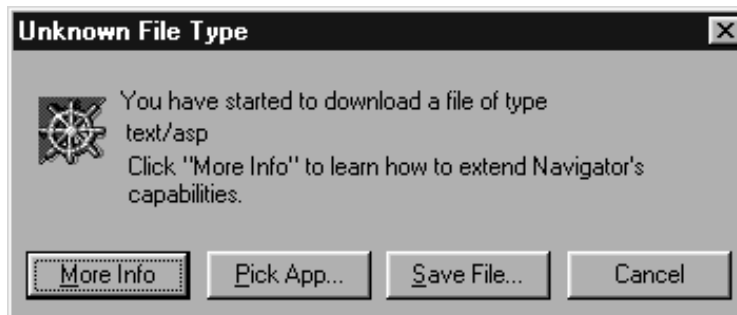
C:\BegASPNET\Ch01\punctual.aspx

Chapter 1

You'll also get this problem if you click on the file in Windows Explorer. If you have Microsoft FrontPage or Visual Studio .NET installed, then it will start up and attempt to help you to edit the code. Otherwise, your browser may display a warning message, or most likely it will ask you which application you wish to use to open up the ASPX file:



Older browsers may try to download the file:



The Problem

This is because you're trying to access the page in a way that doesn't cause the ASP.NET page to be requested **from the web server**. Because you're not requesting the page through the web server, the ASP.NET code doesn't get processed, and that's why you don't get the expected results.

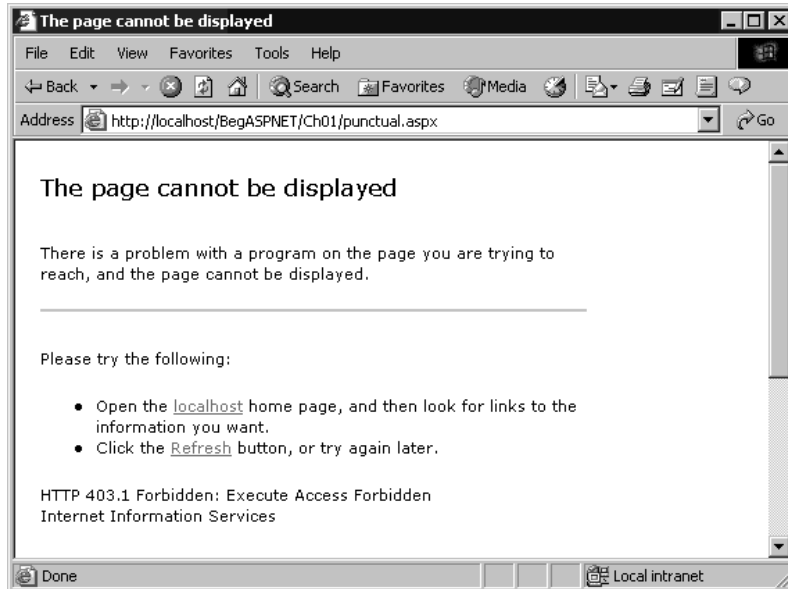
To call the web page through the web server and have the ASP.NET code processed, you need to reference the web server in the URL. Depending on whether you're browsing to the server across a local network, or across the Internet, the URL should look something like one of these:

`http://localhost/BegASPNET/Ch01/punctual.aspx`

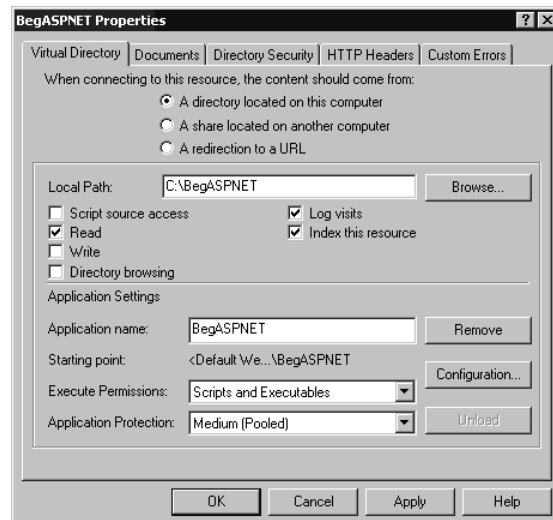
`http://www.distantserver.com/BegASPNET/Ch01/punctual.aspx`

Page Cannot Be Displayed: HTTP Error 403

If you get a 403 error message, then it's probably because you don't have permission to execute the ASP.NET code contained within the page – notice the **Execute Access Forbidden Error** message at the end of the error page:



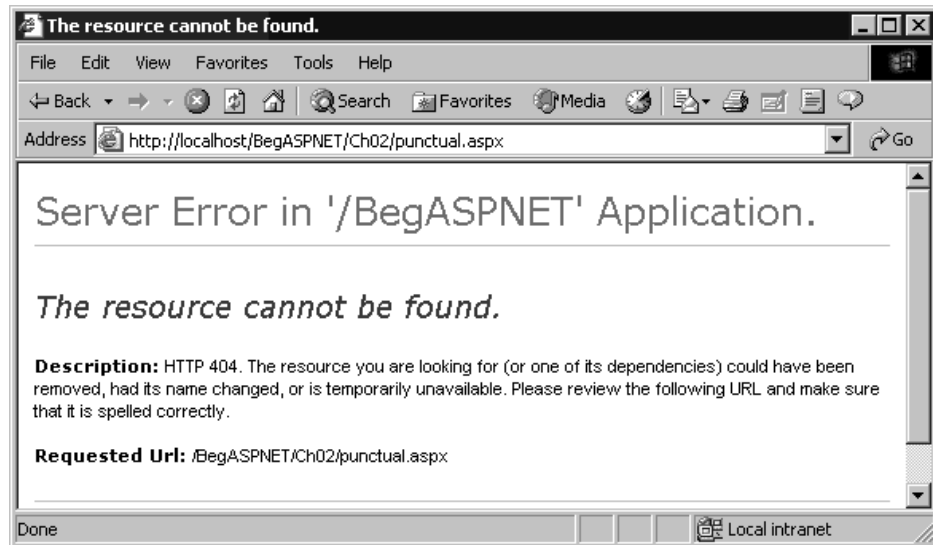
As you'll recall, permissions are controlled by the properties of the virtual directory that contains the ASP.NET page. To change these properties, you'll need to start up the IIS admin snap-in in the MMC, as we described earlier in the chapter. Find the BegASP.NET virtual directory in the left pane, right-click on it, and select Properties. This will bring up the BegASP.NET Properties dialog that we met earlier in the chapter:



Here, you'll need to check that the value shown in the Execute Permissions box is Scripts only or Scripts and Executables –but definitely **not** None.

Page Cannot Be Found: HTTP Error 404

If you get this error message then it means that the browser has managed to connect to the web server successfully, but that the web server can't locate the page you've asked for. This could be because you've mistyped the URL at the browser prompt. In this case, you'll see a message like this:

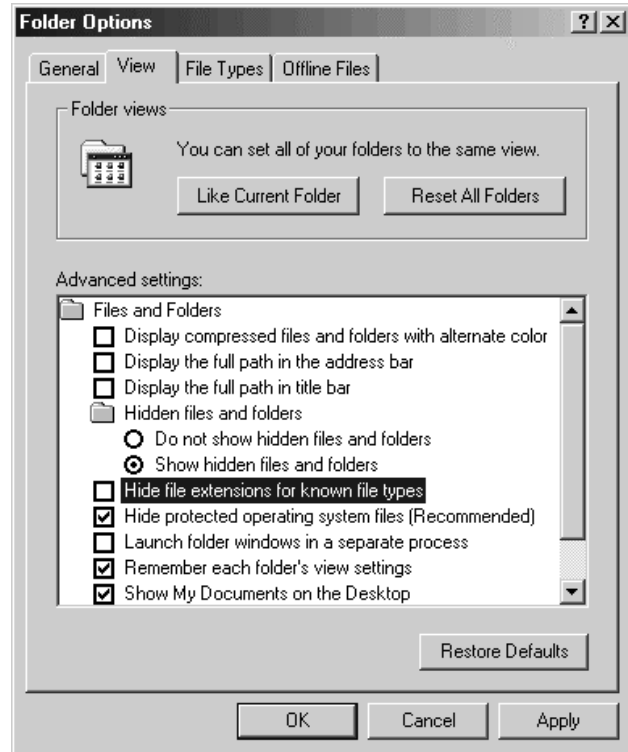


If you get this page, then you might suspect one of the following errors:

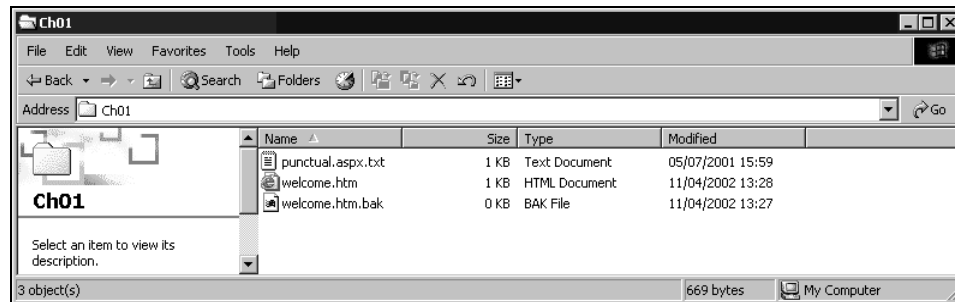
- A simple typing error in the URL, such as `http://localhost/BegASPNET/Ch01/punctually.aspx`
- A wrong directory name, like `http://localhost/BegASPNET/punctual.aspx` instead of `http://localhost/BegASPNET/Ch01/punctual.aspx`
- Including a directory separator (`/`) after the file name, like this `http://localhost/BegASPNET/Ch01/punctual.aspx/`
- Using the directory path in the URL, rather than using the alias, such as `http://chrisu//BegASPNET/Ch01/punctual.aspx`
- Saving the page as an `.html` or `.htmfile`, rather than as an `.aspxfile`, like this `http://localhost/BegASPNET/Ch01/punctual.htm`

Of course, it may be that you've typed in the URL correctly, and you're **still** experiencing this error. In this case, the most likely cause is that you have used Notepad to save your file and that (when you saved the file) it used its default **Save As Type** setting, which is **Text Documents (*.txt)**. This automatically appends a `.txt` suffix to the end of your file name. In this case, you will unwittingly have finished up with a file called `punctual.aspx.txt`.

To check if that is what happened, go to Windows Explorer, and view the (physical) folder that contains the file. Go to the Tools menu and select Folder Options.... Now, in the View tab, ensure that the Hide file extensions for known file types is unchecked, as shown here:



Now click OK and return to view your file in Windows Explorer. You may well see something like the following:



As you can see, Notepad has been less than honest in its dealings with you: when you thought that you had saved your file as `punctual.aspx`, it had inconveniently saved it as `punctual.aspx.txt`. Not surprisingly, your web server won't be able to find your file if it's been renamed accidentally. To correct the filename, right-click on it in the right pane above, select **Rename** from the drop-down menu that appears and remove the `.txt` at the end.

Web Page Unavailable while Offline

Very occasionally, you'll come across the following message box:



This happens because you've tried to request a page and you haven't currently got an active connection to the Internet. This is a misperception by the browser (unless your web server isn't the same machine as the one you're working on) – it is trying to get onto the Internet to get your page when there is no connection, and it's failing to realize that the page you've requested is present on your local machine. One way of retrieving the page is to hit the **Connect** button in the dialog; but that's not the most satisfactory of solutions (since you might incur call charges). Alternatively, you can adjust the settings on your browser. In IE 5/IE 6, select the **File** menu and uncheck the **Work Offline** option.

This could also be caused if you're working on a network and using a proxy server to access the Internet. In this case, you need to bypass the proxy server or disable it for this page, as we described in the section **Browsing to a Page on your Web Server**, earlier in the chapter. Alternatively, if you're using a modem and you don't need to connect, you can correct this misperception by changing the way that IE looks for pages. To do this, select the **Tools | Connections** option and select **Never dial a connection**.

I Just Get a Blank Page

If you see an empty page in your browser then it probably means that you managed to save your `punctual.aspx` without entering any code into it, or that you didn't remember to refresh the browser.

The Page Displays the Message but not the Time

If the web page displays the message "In WebServerLand, the time is currently" – but doesn't display the time – then you might have mistyped the code. For example, you may have mistyped the name of the control:

```
time.text=Hour(Now) & ":" & Minute(Now) & ":" & Second(Now)
```

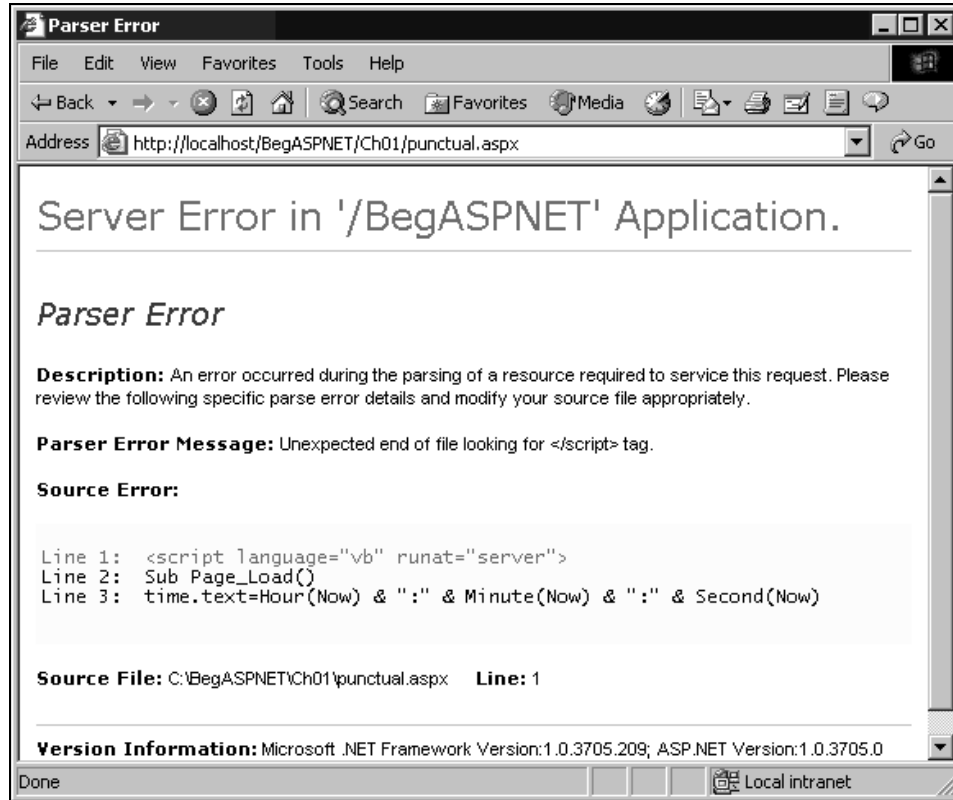
and:

```
<asp:label id="hour" runat="server" />
```

The name of the control 'hour', must match the first word in the line of ASP.NET code, otherwise the control won't be able to identify it.

I Get an Error Statement Citing a Server Error

You may get a message stating that the page cannot be displayed, and citing a server error such as:



This means that there's an error in the ASP.NET code itself. Usually, there's additional information provided with the message. For example, you'd get this error message if you omitted the closing </script> tag on your code. To double-check that this isn't the case, use the sample `punctual.aspx` from the Wrox site at www.wrox.com.

I Have a Different Problem

If your problem isn't covered by this description it's worth testing some of the sample ASP.NET pages that are supplied with the QuickStart tutorials at www.asp.net; (it doesn't matter which installation you've got locally, you can still use these). They should help you to check that IIS has actually installed properly. You can always uninstall and reinstall if necessary, although before you do this, rebooting your server might solve the problem.

You can get support from <http://p2p.wrox.com>, which is our web site dedicated to support issues in this book. Alternatively, there are plenty of other web sites that are dedicated to ASP and ASP.NET. In fact you will find very few sites which focus on just one of the two technologies. Here are just a few:

Chapter 1

<http://www.asp.net>
<http://www.asptoday.com>
<http://www.asp101.com>
<http://www.15seconds.com>
<http://www.4guysfromrolla.com>

There are lots of solutions, discussions and tips on these pages, plus click-throughs to other related pages. Moreover, you can try the newsgroups available on www.asp.net such as [aspngfreeforall](#).

By now, you should have successfully downloaded, set up, and installed both IIS and ASP.NET, and got your first ASP.NET application up and running. If you've done all that, you can pat yourself on the back, make a cup of tea, and get ready to learn about some of the principles behind ASP.NET in the next chapter.

Summary

We started the chapter with a brief introduction to ASP.NET and to dynamic web pages in general and we looked at some of the reasons why you'd want to use a server-side technology for creating web pages. We looked at some of the history behind dynamic web pages, in the form of an overview of the other technologies. We'll expand on this very brief introduction to ASP.NET in the next chapter.

The bulk of the chapter was taken up by a description of the installation process. You *must* have installed IIS 5.0/5.1, MDAC 2.7 *and* either the .NET Framework Redistributable or the .NET Framework SDK to be able to progress further with this book, so please don't be tempted to skip parts that might not have worked. We've listed plenty of resources that will help you get everything up and running, and there's rarely a problem that somebody somewhere hasn't encountered before.

The next chapter covers the software installed with ASP.NET, the .NET Framework, and will build up in much greater detail what ASP.NET does, what the .NET Framework is, and how the two work together.

