Introduction to ASP.NET

ASP.NET is a part of the .NET framework. As a programmer, you interact with it by using the appropriate types in the class library to write programs and design web forms. When a client requests a page, the ASP.NET service runs (inside the CLR environment), executes your code,and creates a final HTML page to send to the client.

To understand ASP.NET's features, it helps to understand ASP's limitations. In other words, before you can understand the .NET solution, you need to understand the problems developersare struggling with today.

Scripting limitations ASP applications rely on VBScript, which suffers from a numberof limitations. To overcome these problems, developers usually need to add separatelydeveloped components, which add a new layer of complexity. In ASP.NET, web pagesare designed in a modern .NET language, not a scripting language.

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

No application structure ASP code is inserted directly into a web page along with HTML markup. The resulting tangle has nothing in common with today's modern, object-oriented languages. As a result, web form code can rarely be reused or modifiedwithout hours of effort.

State limitations One of ASP's strongest features is its integrated session state facility. However, session state is useless in scenarios where a web site is hosted by several separate web servers. In this scenario, a client might access server B while itssession information is trapped on server A and essentially abandoned. ASP.NET corrects this problem by allowing state to be stored in a central repository: either a separate process or a database that all servers can access.

Visual Studio .NET

The last part of .NET is the optional Visual Studio .NET editor, which provides a rich environment where you can rapidly create advanced applications. Some of its features include:

Automatic error detection You could save hours of work when Visual Studio .NET detects and reports an error before you try to run your application. Potential problemsare underlined, just like the "spell-as-you-go" feature found in many word processors.

Debugging tools Visual Studio .NET retains its legendary debugging tools that allowyou to watch your code in action and track the contents of variables.

Page design You can create an attractive page with drag-and-drop ease using VisualStudio .NET's integrated web form designer.

IntelliSense Visual Studio .NET provides statement completion for recognized objects, and automatically lists information such as function parameters in helpfulToolTips.

Basic difference between C# and VB.NET

C# (pronounced "C sharp") and VB.NET (Visual Basic .NET) are both programming languages developed by Microsoft and are part of the .NET framework. While they share many similarities due to their common foundation, there are several differences between them:

1. Syntax:
- C# has syntax that is similar to C and C++, making it familiar to developers from those backgrounds. It follows a C-style syntax with curly braces {} to denote code blocks and uses semicolons to terminate statements.
- VB.NET, on the other hand, has a syntax that is closer to natural language and is often considered more beginner-friendly. It uses keywords like "If", "End If", "For", "Next", etc., and relies on keywords like "Dim" for variable declaration instead of data types.

2. Case Sensitivity:
- C# is case-sensitive, meaning it distinguishes between uppercase and lowercase letters. For example, "myVariable" and "MyVariable" are considered different variables.
- VB.NET, by default, is not case-sensitive. Therefore, "myVariable" and "MyVariable" would refer to the same variable.

3. Development Environments:
- Historically, Visual Studio has been the primary IDE (Integrated Development Environment) for both C# and VB.NET development. However, different versions of Visual Studio may offer different levels of support or features for each language.

4. Historical Background:
- C# was developed by Microsoft as part of its .NET initiative and was released alongside the .NET framework in 2000. It was designed to be a modern, object-oriented language suitable for developing a wide range of applications.
- VB.NET is the successor to Visual Basic, a popular programming language that has been around since the early 1990s. VB.NET was introduced with the .NET framework to provide a more powerful and modernized version of Visual Basic.

5. Community and Adoption:
- C# has generally been more widely adopted and has a larger community of developers compared to VB.NET. This can mean more resources, libraries, and support available for C# developers.
- However, VB.NET still has a significant user base, especially in organizations with a legacy codebase in Visual Basic or where developers have a preference for its syntax or ease of use.

Ultimately, the choice between C# and VB.NET often comes down to personal preference, project requirements, and the existing skill set of the development team. Both languages are capable of building robust and scalable applications on the .NET platform.

6. Syntax and Language Features:
- C# is known for its concise and expressive syntax, which is similar to other C-style languages like C++ and Java. It emphasizes clean, structured code with features such as lambda expressions, LINQ (Language Integrated Query), and asynchronous programming using async/await.
- VB.NET, on the other hand, focuses on readability and ease of use. Its syntax is closer to natural language, making it more approachable for beginners and those transitioning from other non-programming backgrounds. VB.NET includes features like optional parameters, late binding, and support for events with the "Handles" keyword.

Understanding Namespaces and Assemblies

Whether you realize it at first or not, every piece of code in .NET exists inside a class. In turn, every class exists inside a namespace. Figure 3-3 shows this arrangement for your own code andthe DataTime object. Keep in mind that this is an extreme simplification—the System namespace alone is stocked with several hundred classes.

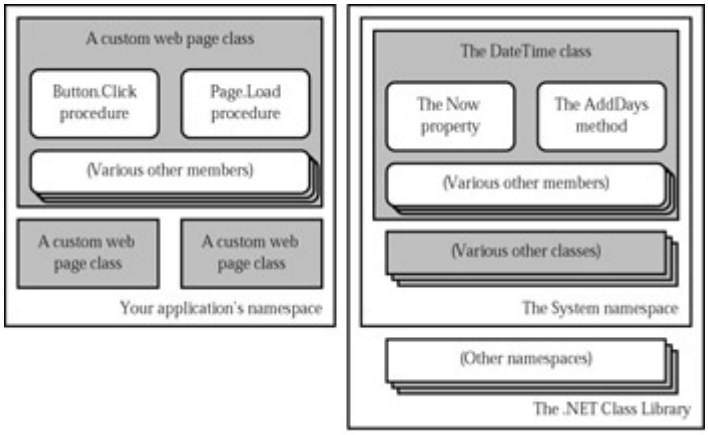


Figure 3-3: .NET namespaces

Namespaces represent the single organizing principle used to group all the different types in the class library. Without namespaces, these types would all be grouped into a single long and messylist, much like the disorganized Windows API. This type of organization is practical for a small set of information, but it would be useless for the thousands of types included with .NET.

Many of the chapters in this book introduce you to one of .NET's namespaces. For example, in thechapters on web controls, you'll learn how to use the objects in the System.Web.UI namespace. In the chapters about Web Services, you'll study the types in the System.Web.Services namespace.For databases, you'll turn to the System.Data namespace. In fact, you've already learned a little about one namespace: the basic System namespace that contains all the simple data types explained in the previous chapter. To continue your exploration after you've finished the book, you'll need to turn to the MSDN reference, which painstakingly documents the properties, methods, and events of every class in every namespace (see Figure 3-4).

Figure 3-4: The MSDN Class Library reference

Often when you write ASP.NET code, you will just use the default namespace. If, however, youwant to create a more specific namespace, you can do so using a simple block structure:

VB .NET

```
Namespace MyCompany
   Namespace MyApp
      Public Class Product'
         Code goes here.
      End Class End
   Namespace
End Namespace
```

C#

```
namespace MyCompany
{
   namespace MyApp
   {
      public class Product
      {
         // Code goes here.
      }
   }
}
```

In the preceding example, the Product class is in the namespace MyCompany.MyApp. Code inside this namespace can access the Product class by name. Code outside it needs to use thefully qualified name, as in MyCompany.MyApp.Product. This ensures that you can use the components from various third-party technology developers without worrying about a name collision. If they follow the recommended naming standards, their classes will always be in a namespace that uses the name of their company and software product. The fully qualified namewill then almost certainly be unique.

Namespaces don't take a private or public access keyword, and can be stacked as many layersdeep as you need.

Importing Namespaces

Having to type long fully qualified names is certain to tire out the fingers and create overly verbosecode. To tighten things up, it's standard practice to import the namespaces you want to use. Whenyou import a namespace, you don't need to type the fully qualified name. Instead, you can use theobject as though it is defined locally.

To import a namespace, you use the Imports statement in VB .NET, or the using statement in C#.These statements must appear as the first lines in your code file, outside of any namespaces or block structures.

VB .NET

```
Imports MyCompany.MyApp
```

C#

```
using MyCompany.MyApp;
```

Consider the situation without importing a namespace:

VB .NET

```
Dim salesProduct As New MyCompany.MyApp.Product()
```

C#

```
MyCompany.MyApp.Product salesProduct = new MyCompany.MyApp.Product();
```

It's much more manageable when you import the MyCompany.MyApp namespace:

VB .NET

```
Dim salesProduct As New Product()
```

C#

```
Product salesProduct = new Product();
```

Assemblies

You might wonder what gives you the ability to use the class library namespaces in a .NET program. Are they hard-wired directly into the language? The truth is that all .NET classes are contained in assemblies, which is the .NET equivalent of traditional .exe and .dll files. There aretwo ways you can use an assembly:

- By placing it in the same directory or in a subdirectory of your application. The CLR will automatically examine all these assemblies and make their classes available to your application. No additional steps or registration are required.

- By placing it in the Global Assembly Cache (GAC), which is a systemwide store of shared assemblies. You won't take this option for your own components unless you are a component vendor. All the .NET classes, however, are a part of the GAC, because it wouldn't make sense to install them separately in every application directory. You'll learnmore about assemblies in Chapter 21.

Assemblies and the .NET Classes

The .NET classes are actually contained in a number of different assemblies. For example, the basic types in the System namespace come from the mscorlib.dll assembly. Many ASP.NET typesare found in the System.Web.dll assembly. If you are precompiling your ASP.NET pages, you will need to explicitly reference these assemblies.

Web Servers and User

Web servers run special software (namely, the built-in Internet Information Services, or IIS) to support mail exchange, FTP and HTTP access, and everything else clients expect in order to access web content. In most cases, you won't be developing on the same computer that you usefor actual web hosting. If you were, you would hamper the performance of your web server by tying it up with development work. You might also frustrate clients when buggy testing leads to crashes and renders the web site unavailable, or when you accidentally overwrite the real deployed web site with a work in progress. Generally, you will perfect your web application on another computer, and then just copy all the files to the web server.

However, in order to use ASP.NET, your computer needs to act like a web server. In fact, while you are testing an ASP.NET application your development computer will work in exactly the sameway as it would over an Internet connection to a remote client. When you test a page, you will actually access the page through IIS (which runs the ASP.NET service), and retrieve the final HTML through an HTTP transfer. The only difference between the way your computer works and the way a web server behaves is that your computer won't be visible and accessible to remote clients on the Internet.

This setup ensures that you can perfect security and perform realistic testing on your developmentcomputer. It also requires you to have the IIS software on your computer in order to use ASP.NET.IIS is included with Windows 2000, Windows XP, and Windows NT, but it is included as an optional component and is not installed by default. To create ASP.NET programs successfully, you need to make sure IIS is installed, preferably before you install the .NET framework or Visual Studio .NET.

Installing IIS

Installing IIS is easy:

1. Click Start, and select Settings | Control Panel.
2. Choose Add Or Remove Programs.
3. Click Add/Remove Windows Components.
4. If Internet Information Services is checked (see Figure 4-1), you already have this

component installed. Otherwise, click it and click Next to install the required IIS files.

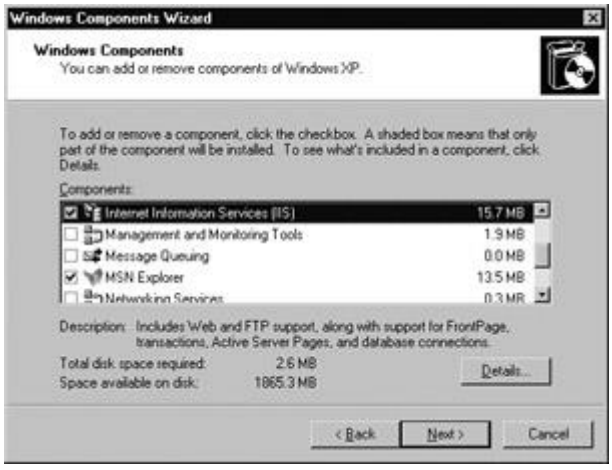


Figure 4-1: Adding the IIS component

When IIS is installed, it automatically creates a directory that represents your web site. This is the directory c:\Inetpub\wwwroot. Any files in this directory will appear as though they are in the root ofyour web server.

To test that IIS is installed correctly and running, browse to the c:\Inetpub\wwwroot directory, and verify that it contains a file called localstart.asp. This is a simple ASP (not ASP.NET) file that is automatically installed with IIS. If you try to double-click on this file to run it from the local directory,you'll receive an error message informing you that your computer doesn't know how to handle direct requests for ASP files (see Figure 4-2), or another program such as Microsoft FrontPage willtry to open it for editing.

Figure 4-2: Attempting to open an ASP file through Windows Explorer

This limitation exists because of the way that ASP and ASP.NET files are processed. In a typical web scenario, IIS receives a request for a file. It then looks at the file extension, and uses that to determine if the file should be allowed, and what program (if any) is required to handle it. OrdinaryHTML files will be sent directly to your browser. ASP files, however, will be processed by the ASPservice (which IIS starts and invokes automatically). The ASP service creates an HTML stream with the final result of its processing, and IIS sends this file to the client. The whole process, whichis diagrammed in Figure 4-3, is quite similar to how ASP.NET files are processed, as you'll see later.

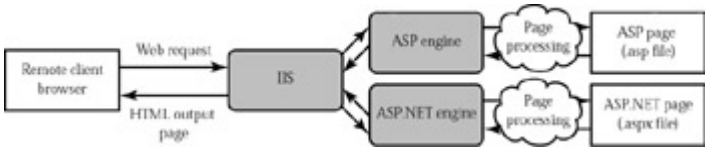


Figure 4-3: How IIS handles an ASP file request

So, in order to test that IIS is working correctly, you need to request the localstart.asp file through IIS (and over the standard HTTP channel). To do this, open an Internet browser, and type a request for the file using the name of your computer. For example, if your computer name is devtest, you would type http://devtest/localstart.asp. IIS will receive your request, ASP will processthe file, and you'll receive a generic introductory page in your browser (see Figure 4-4).

Figure 4-4: Requesting an ASP file through IIS

If you don't know the name of your computer, you can always find it out in IIS Manager. You Can also use one of two presets that automatically refer to the local computer, whatever its name.

These are the "loopback" address 127.0.0.1, and the alias localhost. That means you can try http://localhost/localstart.asp and http://127.0.0.1/localstart.asp.

IIS Manager

To add more ASP pages to your web server, you can copy the corresponding files into the c:\Intetpub\wwroot directory. You can even create subdirectories to group together related resources. However, this makes for poor organization. To properly use ASP or ASP.NET, you need to make your own custom virtual directory for each application you are designing. With a virtual directory, you can expose any physical directory (on any drive on your computer) on yourweb server, as though it were located in the c:\Inetpup\wwwroot directory.

To create virtual directories, you need to use the administrative IIS Manager program. To start it,select Settings | Control Panel | Administrative Tools | Internet Services Manager from the Start menu on the taskbar.

Creating a Virtual Directory

To create a new virtual directory for an existing physical directory, right-click on the Default WebSite item (under your computer in the tree), and choose New | Virtual Directory from the contextmenu. A wizard will start, as shown in Figure 4-6.

Figure 4-6: The Virtual Directory Creation Wizard

As you step through the wizard, you will need to provide several pieces of information.

Alias

The alias is the name a client will use to access the files in this virtual directory. For example, ifyour alias is MyApp and your computer is MyServer, the user will request pages like http://MyServer/MyApp/MyPage.aspx.

Directory

The directory is the physical source on your hard drive that is exposed as a virtual directory. For example, c:\Intetpub\wwwroot is the physical directory that is used for the root virtual directory of your web server. IIS will provide access to all the allowed file types in this directory. Generally, when starting a web application you will create a new physical directory to use, and then map it toa suitable virtual directory name.

Permissions

Finally, you're asked to set Permissions for your virtual directory (Figure 4-7). There are severaloptions:

- **Read** is the most basic permission—it's required in order for IIS to provide any requested files to the user. If this is disabled, the client will not be able to access ASP or ASP.NET pages, or static files like HTML and images. Note that even when you enable read permission, there are several other layers of possible security in IIS. For example, some file types (such as those that correspond to ASP.NET configuration files) are automatically restricted, even if they are in a directory that has read permission.

- **Run scripts** allows the user to request an ASP or ASP.NET page. If you enable read, but don't allow script permission, the user will be restricted to static file types such as HTML documents. ASP and ASP.NET pages require a higher permission because they could conceivable perform operations that would damage the web server or compromise security.

- **Execute** allows the user to run an ordinary executable file or CGI application. This is a possible security risk as well, and shouldn't be enabled unless you require it (which youwon't for ordinary ASP or ASP.NET applications).

- **Write** allows the user to add, modify, or delete files on the web server. This permission should never be granted, because it could easily allow the computer to upload and then execute a dangerous script file (or at the least, use up all your available disk space). Instead, use an FTP site, or create an ASP.NET application that allows the user to upload specifictypes of information or files.

- **Browse** allows the user to look at all the files in the virtual directory, even if the contents of those files are restricted (as in the case of a special ASP.NET configuration file). Browse is generally disabled, because it allows users to discover additional information about your website and its structure, and exploit possible security holes. On the other hand, it is quite usefulfor testing on a development computer.

Figure 4-7: Virtual directory permissions

You only need to enable the first two checkboxes, although you can allow more on a developmentcomputer that will never act as a live web server (keep in mind, however, this could allow other users on a local network to access and modify your files in the virtual directory). You can also change these settings after you have created the virtual directory.

Virtual Directories and Applications

You can manage all the virtual directories on your computer in IIS manager by expanding the list under Default Web Site. You'll notice that items in the tree have three different types of icons (see Figure 4-8).

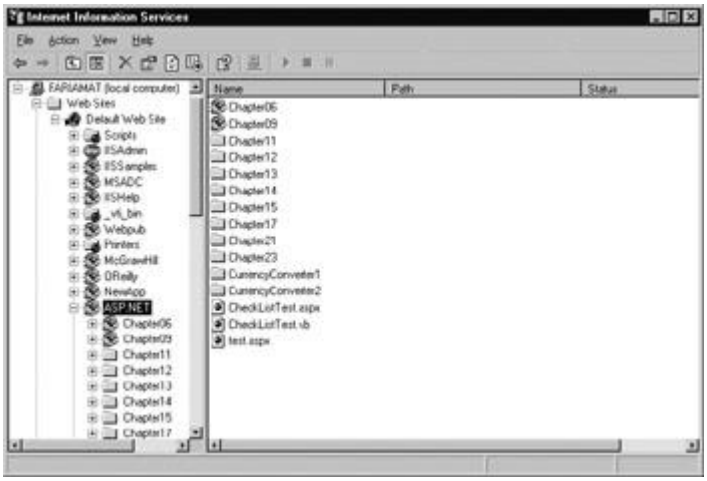


Figure 4-8: Web folders

Ordinary folder This represents a subdirectory inside another virtual directory. For example, if you create a virtual directory and then add a subdirectory to the physicaldirectory, it will be displayed here.

Folders with a globe This represents a distinct virtual directory.

Package folders This represents a virtual directory that is also a web application. By default, when you use the wizard to create a virtual directory, it is also configured as aweb application. This means that it will share a common set of resources and run in itsown separate application domain. The topic of web applications is examined in more detail in Chapter 5.

Virtual Directories Allow Access to Subdirectories

Imagine you create a virtual directory called MyApp on a computer called MyServer. The virtual directory corresponds to the physical directory c:\MyApp. If you add the subdirectory c:\MyApp\MoreFiles, this directory will automatically be included in the IIS Manager list as an ordinary folder. Clients will be able to access files in this folder by specifying the folder name, as inhttp://MyServer/MyApp/MoreFiles/.

By default, the subdirectory will inherit all the permissions of the virtual directory. However, you can change these settings in IIS Manager. This is a common technique used to break a single application into different parts (for example, if some pages require heightened security settings).

Folder Settings

Permissions are managed on a directory-by-directory basis, which means that all the files in a directory

need to share the same set of permissions. (Of course, not all file types are treated equally, as you'll see in the next chapter.) To apply different settings, you would typically create avirtual directory with more than one subdirectory, and set different options for each subdirectory.

To configure the options for a directory, right-click on it and choose Properties. The Propertieswindow will appear, as shown in Figure 4-9.

Figure 4-9: Web directory properties

The Virtual Directory tab contains the most important settings. These include options that allow you to change whatever permissions you set in the wizard. You are also provided with the local path that this directory corresponds to. If it's a virtual directory, you can change it to correspond toa different physical directory, but if it is an ordinary subdirectory inside a virtual directory, this fieldwill be read-only.

Remember, when you create a virtual directory with the wizard, it is also configured as a web application. You can change this by clicking the Remove button next to the application name. Similarly, you can click the Create button to transform an ordinary virtual directory into a full- fledged application. Usually you won't need to perform these tasks, but it's nice to know they areavailable if you need to make a change.

Changes that you make are automatically propagated down to all subdirectories. If you want to make a change that will affect all applications, you can do it from the root directory (by right- clicking on the Default Web Site item and choosing Properties). If you have set custom settings inany other application or subdirectory, you will be warned. IIS will then present a list of directories that will be affected, as shown in Figure 4-10, and give you the chance to specify exactly which

ones you want to change and which ones you want to leave as is.

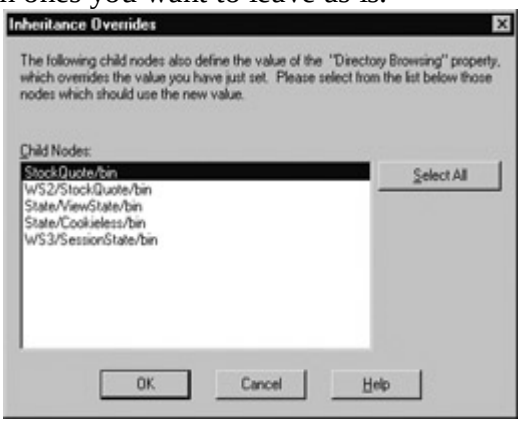


Figure 4-10: A change that affects several directories

If you explore the directory properties, you'll discover several other settings that affect ASP applications but don't have any effect on ASP.NET sites. These include the Application Protection(memory isolation) setting, and the options for session state, script timeout, and default language.These settings are all replaced by ASP.NET's new configuration file system, which you'll learn about in the next chapter. Some other settings are designed for security, and these are examined in Chapter 24.

Adding a Virtual Directory to Your Neighborhood

Working with a web application can sometimes be a little inflexible. If you use Windows Explorerand look at the physical directory for the web site, you can see the full list of files, but you can't execute any of them directly. On the other hand, if you use your browser and go through the virtual directory, you can run any page, but there's no way to browse through a directory listing because virtual directories almost always have directory browsing permission disabled.

While you're developing an application you may want to circumvent this limitation. That way you can examine exactly what comprises your web application, and run several different pages easily,without needing to constantly type a full filename or dart back and forth between Internet Explorerand Windows Explorer. All you need to do is enable directory browsing for your virtual directory inIIS Manager. (You can easily enable or disable this setting from the directory properties window.)

Next, to make life even easier, you can add the virtual directory to the network neighborhood in Windows Explorer. First, open Windows Explorer. Then click on Network, and double-click on AddNetwork Place from the file list, as shown in Figure 4-13.

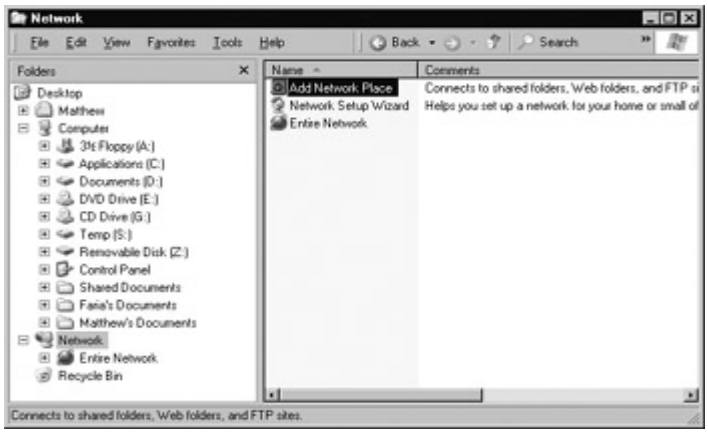


Figure 4-13: Select Add Network Place

The Add Network Place Wizard will appear. This wizard allows you to create a network folder in Windows Explorer that represents your virtual directory (see Figure 4-14). The only important piece of information you need to specify is the address for your virtual directory (such as http://MyServer/MyFolder). Don't use the physical directory path.

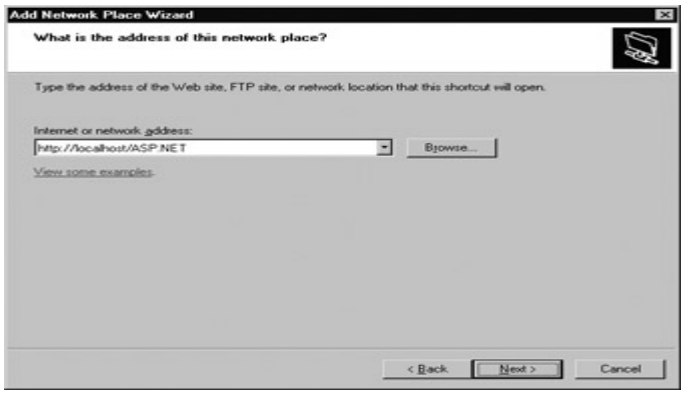


Figure 4-14: Specify the virtual directory

You can then choose the name that will be used for this virtual directory. Once you finish the wizard, the directory will appear in your network neighborhood, and you can browse through theremote files (see Figure 4-15). The interesting thing is that when you browse this directory, you are actually receiving all the information you need over HTTP. You can also execute an ASP or ASP.NET file by double-clicking—which you can't do directly from the physical directory.

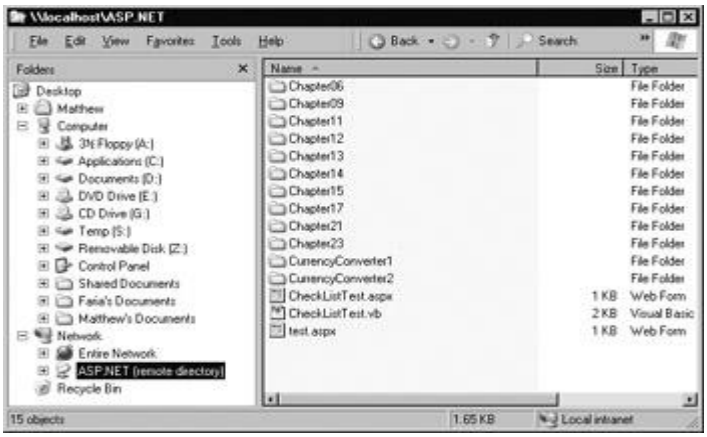


Figure 4-15: The mapped virtual directory

Installing ASP.NET

Once you've installed IIS, you can install ASP.NET. At the time of publication ASP.NET was available in two different packages, although Microsoft is certain to add even more options.

- The free .NET redistributable.
- The Visual Studio .NET application, which includes the .NET framework and an IDE todesign your applications.
- The ASP.NET Premium edition, which can be installed instead of the standard redistributable, or after it. This edition adds support for output caching, web farm sessionstate, and more than four CPUs in the web server.

Like most Microsoft setups, the process is quite straightforward. A graphical wizard takes youthrough every step of the extremely long process (see Figure 4-16).

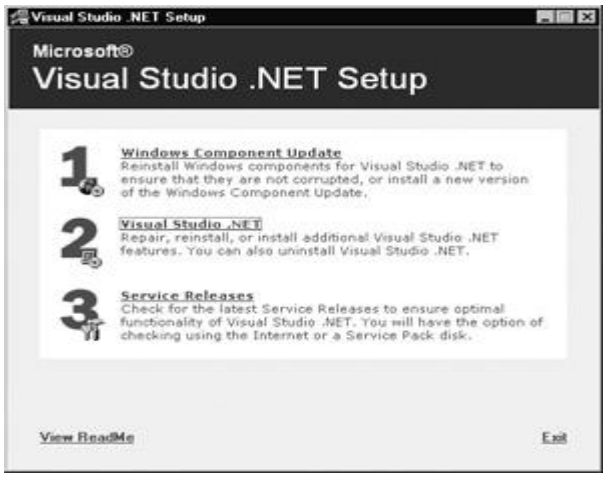


Figure 4-16: The Visual Studio .NET setup

Depending on the version of the setup that you are using, the Microsoft Data Access Components may not be automatically installed. If this is the case, you will receive a warning message when you start the install that asks if you want to continue. Before going any further, you shouldinstall these components. You can find them on the Internet at http://www.microsoft.com/data/.

2.7

ASP.NET Applications

Overview

The last few chapters gave a sweeping introduction to the .NET framework, including its languages, architecture, and underlying philosophy. They also examined how to set up your computer with the software and virtual directories you need to make ASP.NET applications. In thischapter, we dive into real ASP.NET programming with our first .aspx page.

As we introduce ASP.NET applications, you'll learn some of the core topics that every ASP.NET developer must master. We'll examine what makes up an ASP.NET site, and what file types you can use. Even more importantly, you'll learn how an ASP.NET application works behind the scenes, how to configure it, and how to use code-behind development to separate your web pageuser interface from its business logic. These concepts are the starting point for any ASP.NET programming project.

ASP.NET Applications

It's sometimes difficult to define exactly what a web application is. Unlike a traditional desktop program (which is usually contained in a single .exe file), ASP.NET applications are divided into multiple web pages. This division means that a user can often enter an ASP.NET application at several different points, and follow a link out of your application to another part of your web site oranother web server. So does it make sense to consider a web site as an application?

In ASP.NET, the answer is yes. Every individual ASP.NET application shares a common set of resources and configuration options. Web pages from other ASP.NET applications, even if they are on the same server, do not share these resources. Technically speaking, every ASP.NET application is executed inside a separate application domain, which is roughly similar to a Windows "process" in ordinary unmanaged code. Application domains are isolated in memory, meaning that if one web application causes a fatal error it won't affect any other applications thatare currently running. Similarly, it also means that the web pages in one application can't accessthe in-memory information from another application. Each application has its own set of caching,application, and session state data.

The standard definition of an ASP.NET application describes it as a combination of files, pages, handlers, modules, and executable code that can be invoked from a virtual directory (and, optionally, its subdirectories) on a web server (see Figure 5-1). In other words, the virtual directoryis the basic grouping structure that delimits an application.

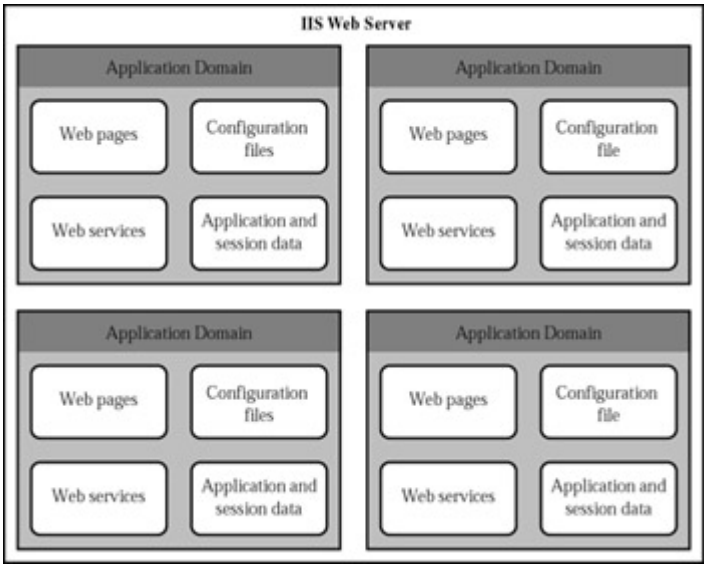


Figure 5-1: ASP.NET applications

In Chapter 4, you saw how you could create a virtual directory. The next step is to examine whatthis directory can contain.

ASP.NET File Types

Applications in ASP.NET are a little more sophisticated than their ASP counterparts, and supportseveral different types of files. Table 5-1 describes these file types.

Table 5-1: ASP.NET File Types

File Name	Description
Ends with .aspx	These are ASP.NET web pages (the .NET equivalent for the .asp file). They contain the user interface and some or all of the underlyingcode. Users request or navigate directly to one of these pages to start your web application.
Ends with .ascx	These are ASP.NET user controls. They are very similar to web pages, except that they must be hosted inside an .aspx file. User controls allow you to develop an important piece of user interface andreuse it in as many web forms as you want without repetitive code.
Ends with .asmx	These are ASP.NET Web Services, which are described in the fourthpart of this book. Web Services work differently than web pages, but they still share the same application resources, configuration settings, and memory.
web.config	This is the XML-based configuration file for your ASP.NET application. It includes settings for customizing security, state management, memory management, and much more.
global.asax	This is the global application file (the .NET equivalent for the global.asa file). You use this file to define global variables and reactto global events.
Ends with .disco or .vsdisco	These are special "discovery" files used to help clients find Web Services. You'll discover more about them in the fourth part of thisbook.
Ends with .vb or .cs	These are code-behind files created in Visual Basic or C#. They allow you to separate code from the user interface logic in the web form page. I use code-behind extensively in this book, and introduceit in this chapter.
Ends with .resx	These files may exist if you are using Visual Studio .NET. They areused to store information that you add at design time.
Ends with .sln, .suo, and .vbproj, csproj	These files are used by Visual Studio .NET to group together projects(a collection of files in a web application) and solutions (a collection of projects that you are developing or testing at once). They store a list of related files and some options for the Visual Studio .NET IDE. These files are only used during development and should not be deployed to a web server. However, even if they are the default, ASP.NET security setting will prevent a user from viewing them.

In addition, your directory can contain other resources that aren't special ASP.NET files. For example, it could hold image files, HTML files, or cascading style sheets (CSS files). Visual Studio .NET even adds a Styles.css file to your projects automatically for you to define styles thatyou want to use with controls. Its use is completely optional, and it has more to do with straight HTML than ASP.NET programming.

All of these files types are optional. You can create a legitimate ASP.NET application with a singleweb form (.aspx file) or Web Service (.asmx file).

The bin Directory

In addition, every web application directory can have a special subdirectory called \bin. This directory holds the .NET assemblies used by your application. For example, you can develop a special database component in any .NET language, compile it to a DLL file, and place it in this directory. ASP.NET will automatically detect it and allow any page in that application to use it. Thisis far easier than traditional COM-based component development, which requires a component to be registered before it can be used (and often re-registered when it changes). Component-based development is explored in Chapter 21.

The \bin directory is also used with Visual Studio .NET to hold a compiled version of your code. This topic, which is a common source of confusion for new ASP.NET developers,

Code-Behind

Before we can finish our transformation from traditional ASP development to the object- oriented .NET world, we need to consider the technique of code-behind programming. This innovation is so important to ASP.NET development that I believe all professional ASP.NETdevelopers will adopt it eventually. Microsoft-sponsored best-of-breed platform samples (forexample, the IBuySpy case studies we'll explore in Chapter 25) use it extensively. Visual Studio .NET, a premier tool for creating ASP.NET sites, uses it natively and exclusively.

With code-behind, you create a separate code file (a .vb file for VB .NET or .cs file for C#) to match every .aspx file. The .aspx file contains the user interface logic, which is the series of HTMLcode and ASP.NET tags that create controls on the page. The .vb or .cs file contains code only.

To create the .aspx file, you can use the advanced features of any HTML editor, or you can drag- and-drop your way to success with Visual Studio .NET's own integrated web form designer, whichhas many more indispensable features. At the top of the .aspx file, you use a special Page directive that identifies the matching code-behind file.

```
<%@ Page Language="VB" Inherits="MyPageClass" Src="MyPage.vb" %>
```

This directive defines the language (VB), identifies the page class that contains all the page code (MyPageClass), and identifies the source file where the page class can be found (MyPage.vb). Like the .aspx file, the source code is contained in a plain text file. ASP.NET will compile it automatically for a web request.

The Page Directive Sets Pre-processor Options

Directives are processed before any ASP.NET code is executed, and set various options that influence how the ASP.NET compiler processes your file. Setting the default language and the code-behind file are only two possibilities. You can also add attributes that will configure tracing, language options, and state management settings. These attributes are explained throughout thisbook, in the related chapters.

The MyPage.vb file needs to have a page class, which is defined like any other .NET class. InVisual Basic, you use a Class/End Class block.

```
Public Class MyPageClass
  Inherits System.Web.UI.Page'
  (Code goes here.)
End Class
```

The only trick is that this page inherits from a special generic Page class. If you rememberthe .NET primer in Chapter 3, you'll recognize that the Page class is in the namespace System.Web.UI, which contains all sorts of types used for ASP.NET user interface.

You can now add code inside the page class. For example, you could rewrite HelloWorld2.aspxwith code-behind as two files:

HelloWorld2.aspx

```
<%@ Page Language="VB" Inherits="HelloWorldClass"
Src="HelloWorld2.vb" %>
<html>
<body>
<form id="Form" runat="server">
    <asp:Label id="lblTest" runat="server" />
</form>

</body>
</html>
```

HelloWorld2.vb

```
Public Class HelloWorldClass
  Inherits System.Web.UI.Page
    Protected lblTest As System.Web.UI.WebControls.Label

    Public Sub Page_Load()
        lblTest.Text = "Hello, the Page_Load event occurred."End
    Sub

End Class
```

In the HelloWorld2.vb code-behind file, you need to add an additional line to define the Labelcontrol. All ASP.NET controls must be defined using the Protected keyword.

```
Protected lblTest As System.Web.UI.WebControls.Label
```

You should make sure that the control name matches the id attribute in the <asp:Label> tag exactly. This allows ASP.NET to connect the controls in your .aspx template file to the controls that you manipulate in your code. It also allows code editors like Visual Studio .NET to check for syntax errors, such as an attempt to use an undeclared variable or a property that the label controldoesn't support. You'll learn more about how Visual Studio .NET automates some of the chores associated with ASP.NET development in Chapter 8.

In the case of HelloWorld.aspx, the code is so simple that you don't receive much of a benefit fromcode-behind development. However, the advantage increases dramatically as you begin to add more complex ASP.NET controls and code routines.

Global.asax Code-Behind

The global.asax file also supports code-behind development. In this case, however, the code- behind class does not inherit from the Page class, because the global.asax file does not representa page. Instead, it inherits from the special System.Web.HttpApplication class. Similarly, the global.asax uses an application directive instead of a Page directive.

The next example rewrites our global.asax file using code-behind programming.

global.asax

```
<%@ Application Codebehind="Global.vb" Inherits="GlobalApp" %>
```

global.vb

```
Public Class Global
  Inherits System.Web.HttpApplication

    Public Sub Application_OnEndRequest()
        Response.Write("This page was served at " & _
        DateTime.Now.ToString())
    End Sub

End Class
```

Once again, ASP.NET always create a custom HttpApplication class to represent your applicationwhen you first run it, as described in the "Behind the Scenes with HelloWorld.aspx" list of steps earlier in this chapter. The difference with code-behind development is that you need to understand and specify that detail explicitly.

Application Events

Application_OnEndRequest is only one of more than a dozen events you can respond to in your code. To create a different event handler, just create a subroutine with the defined name. Table 5- 3 shows some of the most commonly used events.

Table 5-3: Basic Events

Event Name	Description
Application_OnStart	Occurs when the application starts, which is the first time it receives a request from any user. It does not occur on subsequent requests. This event is commonly used to createor cache some initial information that will be reused later.
Application_OnEnd	Occurs when the application is shutting down, generally because the web server is being restarted. You can createcleanup code here.
Application_OnBeginRequest	Occurs with each request the application receives, just beforethe page code is executed.
Application_OnEndRequest	Occurs with each request the application receives, just afterthe page code is executed.
Session_OnStart	Occurs whenever a new user request is received and a session is started. Sessions are described in Chapter 10.
Session_OnEnd	Occurs when a session times out or is programmatically ended.
Application_OnError	Occurs when an unhandled error occurs. You can find more information about error handling in Chapter 11.

Understanding ASP.NET Classes

Once you understand that all web pages are instances of the Page class and all global.asax filesare instances of the HttpApplication class, many mysteries are cleared up.

For example, consider this line from the global.asax (or global.vb) file:

```
Response.Write("This page was served at " & DateTime.Now.ToString())
```

You might wonder what gives you the ability to use the Response object. In previous versions of ASP, Response was just a special object built into ASP programming. ASP.NET, however, uses amore consistent object-oriented approach. In ASP.NET, the Response object is available becauseit's a part of the HttpApplication class.

When you inherit from HttpApplication (either explicitly in a code-behind file or automatically), yourclass acquires all the features of the HttpApplication class. These include properties such as Request, Response, Session, Application, and Server. In turn, each of these properties holds an instance of a special ASP.NET object. For example, HttpApplication.Response holds a reference to the HttpResponse object, which provides a method called Write.

Similarly, when you create an .aspx page, you are creating a class that inherits from ASP.NET'sPage class. This class also provides properties such as Request, Response, Session, and Application, along with a number of additional user-interface related members.

The Built-In ASP Objects Are Still Available

ASP.NET provides the traditional built-in objects through properties in basic classes such as Pageand Application. For backward compatibility, these objects still support almost all the ASP syntax, and they add some new properties. However, the major change between ASP and ASP.NET is that these built-in objects are used much less often. For example, dynamic pages are usually created by setting control properties rather than using the Reponse.Write command. These controls are introduced in Chapter 6.

The MSDN Reference

Understanding the underlying object model is probably the best way to become an ASP.NET guruand distinguish yourself from other, less experienced ASP.NET programmers. It allows you to usethe MSDN to find out the full set of capabilities available to you. For example, now that you understand all global.asax files are custom HttpApplication instances, you can refer to the documentation for the HttpApplication class (see Figure 5-12). You can use any of the HttpApplication properties inside your global.asax or global.vb file. Similarly, you can write event handlers for any of its events.

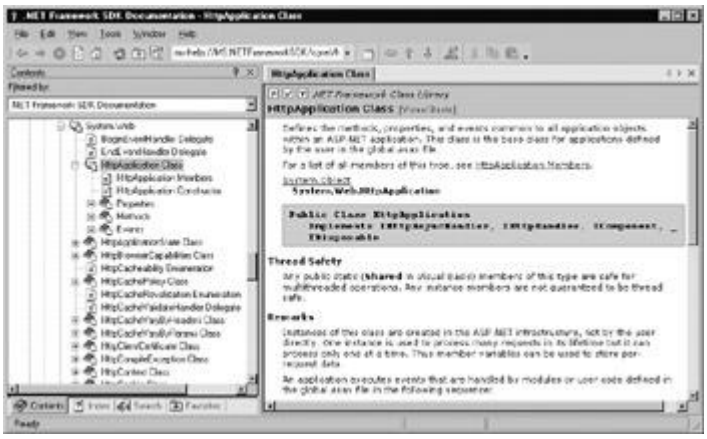


Figure 5-12: The MSDN Help for HttpApplication

If you also understand that the Response property is an instance of the HttpResponse class, youcan find out more information about the ways you can use the Response object.

The basic set of ASP.NET objects is described in the next chapter. However, knowing the classesthat these objects are based on allows you to get a better understanding of ASP.NET development.

ASP.NET Configuration

The last topic we'll consider in this chapter is the web.config configuration file. In the MyFirstWebApp application, there is currently no web.config file. That means that the defaultsettings are used from the server's machine.config file.

Every web server has a single machine.config file that defines the default options. Typically, this file is found in the directory C:\WINNT\Microsoft.NET\Framework\[version]\CONFIG, where [version] is the version number of the .NET framework. Generally, you won't edit this file manually.It contains a complex mess of computer-specific settings. Instead, you can create a web.config filefor your application that contains all the special settings.

The web.config files has several advantages over traditional ASP configuration:

It's never locked. As described in the beginning of this chapter, you can updateweb.config settings at any point, and ASP.NET will smoothly transition to a new application domain.

It's easily accessed and replicated. With the appropriate rights, you can change a web.config file from a remote computer. You can also copy the web.config file and useit to apply identical settings to another application, or another web server that runs thesame application in a web farm scenario.

It's easy to edit and understand. The settings in the web.config file are human- readable. In the future, it's likely that Microsoft will provide a graphical tool that automates web.config changes. Even without it, you can easily add or modify settingsusing a text editor like Notepad.

IIS Manager Is Still Required for Some Tasks

With ASP.NET, you don't need to worry about the metabase. However, there are still a few tasksthat you can't perform with a web.config file. For example, you can't create or remove a virtual directory. Similarly, you can't change file mappings. If you want the ASP.NET service to process

requests for additional file types (such as .html, or a custom file type you define suchas .mycompany), you must use IIS Manager, as described in Chapter 4.

The Web.config File

The web.config file uses a special XML format. Everything is nested in a root <configuration> element, which contains a <system.web> element. Inside the <system.web> element are separateelements for each aspect of configuration.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <!-- Configuration sections go here. -->
  </system.web>
</configuration>
```

You can include as few or as many configuration sections as you want. For example, if you needto specify special error settings, you could add just the <customError> group. If you create an ASP.NET application in Visual Studio .NET, a web.config file is created for you with a basic skeleton showing you all the important sections. Additional comments are added to each section,describing the purpose of various options.

```
<!-- This is the format for an XML comment. -->
```

The following example shows a web.config file with its most important sections (and no settings).Note that the web.config file is case-sensitive, and uses a standard where the first word is alwayslowercase. That means you cannot write <CustomErrors> instead of <customErrors>.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <httpRuntime />
    <pages />
    <compilation />
    <customErrors />
    <authentication />
    <authorization />
    <identity />
    <trace />
    <sessionState />
    <httpHandlers />
    <httpModules />
    <globalization />
    <compilation />

  </system.web>
</configuration>
```

If you want to learn about all the settings that are available in the web.config file, you have twooptions:

- In this book, each section discusses any appropriate web.config settings. For example, inChapter 10, the settings in the <sessionState> group are described.
- In Chapter 28, you can find a detailed list of configuration settings.

Nested Configuration

ASP.NET uses a multi-layered configuration system that allows you to use different settings for different parts of your application. To use this technique, you need to create additional subdirectories inside your virtual directory. These subdirectories can contain their own web.configfiles with additional settings.

Subdirectories also inherit web.config settings from the parent directory. For example, consider the web request http://localhost/MyFirstWebApp/Special/SpecialHelloWorld.aspx. which accessesa file in Special, which is a subdirectory of the application virtual directory MyFirstWebApp. The corresponding physical directory is C:\TestWeb.

The web request for SpecialHelloWorld.aspx can acquire settings from three different files, asshown in Figure 5-13.

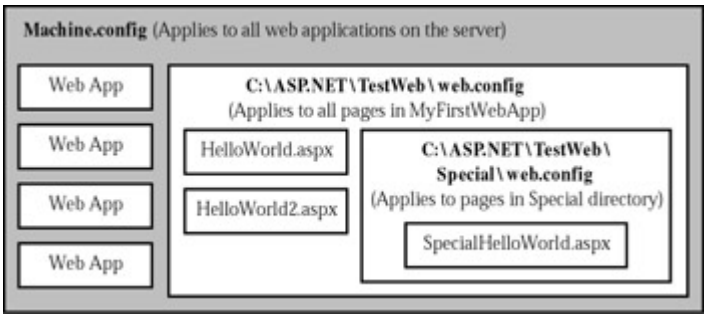


Figure 5-13: Configuration inheritance

Any machine.config or TestWeb\web.config settings that are not explicitly overridden in the TestWeb\Special\web.config file will still apply to the SpecialHelloWorld.aspx page. In this way, subdirectories can specify just a small set of settings that differ from the rest of the web application. One reason you might want to use multiple directories in an application is to apply different security settings. Files that need to be secured would then be placed in a special directory with a web.config file that defines more stringent security settings.

Storing Custom Settings in the Web.config File

Before rounding up our discussion of configuration, it's worth examining how you can add some ofyour own settings to the web.config file, and use them with the simple HelloWorld.aspx page.

You add custom settings to a special element called <appSettings>. Note that this is nested in theroot <configuration> element, not the <system.web> element, which contains the other groups of predefined settings.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <appSettings>
    <!-- Special application settings go here. -->
  </appSettings>

  <system.web>
    <!-- Configuration sections go here. -->
  </system.web>

</configuration>
```

The custom settings that you add are created as simple string variables. There are severalreasons that you might want to use a special web.config setting:

- To make it easy to quickly switch between different modes of operation. For example, youmight create a special debugging variable. Your web pages could check for this variable,

and if it is set to a specified value, output additional information to help you test the application.

- To centralize an important setting that needs to be used in many different pages. For example, you could create a variable that contains a database connection string. Any pagethat needs to connect to the database could then retrieve this value and use it. As you'll learn later in this book, it's a good idea to always use the exact same database connection string in all your pages, as this ensures that SQL Server can reuse connections for differentclients. The technical term for this time-saving feature is connection pooling.

- To set some initial values. Depending on the operation, the user might be able to modifythese values, but the web.config file could supply the default selection.

Custom settings are entered using an <add> element that identifies a unique variable name (key)and the variable contents (value). The following example adds two special variables, one that contains a database connection string, and one that defines a suitable SQL statement for retrieving sales records.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <appSettings>
    <add key="ConnectionString"
        value="Data Source=localhost;Initial
        Catalog=Pubs;User ID=sa" />
    <add key="SelectSales" value="SELECT * FROM Sales" />
  </appSettings>

  <system.web>
    <!-- Configuration sections go here. -->
  </system.web>

</configuration>
```

You can create a simple ShowSettings.vb (or ShowSettings.aspx) file to query this information anddisplay the results. You access these settings by key name, using the System.Configuration.ConfigurationSettings class. This class provides a shared property called AppSettings.

```
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Configuration

Public Class ShowSettings
  Inherits Page
    Protected lblTest As Label

    Public Sub Page_Load()
        lblTest.Text = "This app will connect with the connection
        lblTest.Text &= "string:<br><b>"& _
        ConfigurationSettings.AppSettings("ConnectionString")
        lblTest.Text &= "</b><br><br>"
        lblTest.Text &= "And will execute the SQL Statement:"
        lblTest.Text &= "<br><b>" & _
        ConfigurationSettings.AppSettings("SelectSales")
        lblTest.Text &= "</b>"
    End Sub

End Class
```

Ineresting Facts About this Code

- The System.Configuration namespace is imported to make it easier to access the ConfigurationSettings class.
- The &= operator is used to quickly add information to the label. This is equivalent to writing lblTest.Text = lblText.Text & "<extra content>".
- A few HTML tags are added to the label, including bold tags (<b>) to emphasize certainwords, and a line break (
) to split the output over multiple lines.

Later, in the third part of this book, you'll learn how to use connection strings and SQL statementswith a database. For now, our simple application just displays the custom web.config settings, as shown in Figure 5-14.

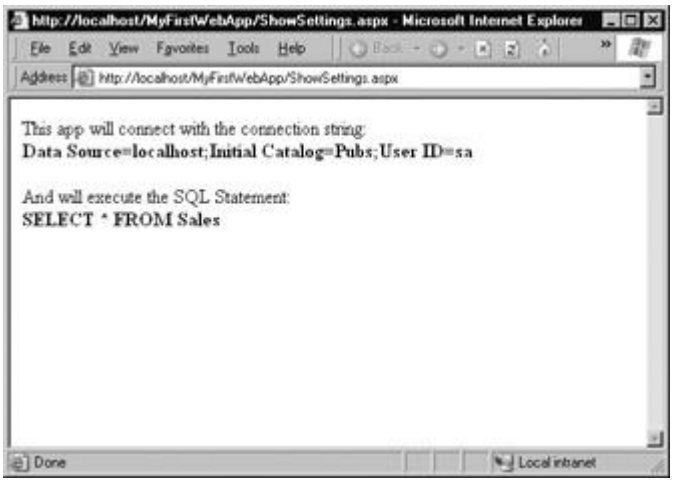


Figure 5-14: ShowSettings.aspx displays the custom application settings

Isn't This a Security Risk?

Unlike code files, the web.config can't be deployed in a compiled form. For that reason, it might seem like a potential security risk to store information such as a database access password in plain text. Indeed, in some cases you might want to use precompiled code-behind files or other measures to make sure information like this is harder to reach. However, ASP.NET is configured,by default, to deny any requests for .config files. That means a remote user will not be able to access the file through IIS. Instead, they'll receive the error message shown in Figure 5-15.

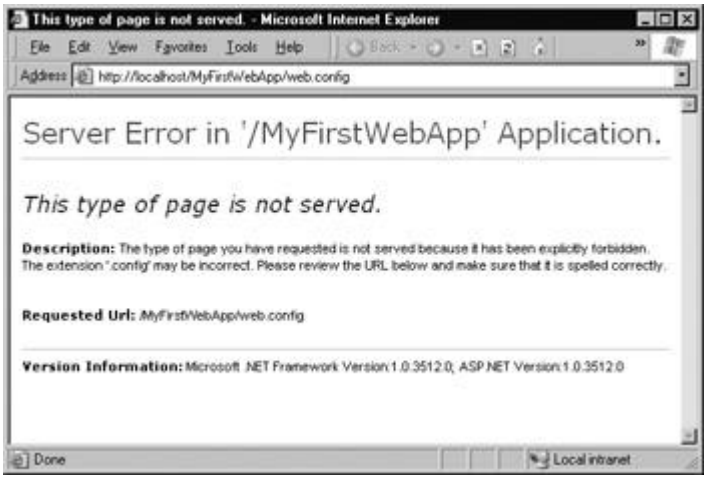


Figure 5-15: Requests for web.config are denied

Web Form Fundamentals

Overview

ASP.NET introduces a remarkable new model for creating web pages. In traditional ASP development, programmers had to master the quirks and details of HTML markup before being able to design dynamic web pages. Pages had to be carefully tailored to a specific task, and additional content could only be generated by writing raw HTML tags. Worst of all, ASP.NET programs usually required extensive code changes whenever a site's user interface was updatedwith a new look.

In ASP.NET, you can use a higher-level model of server-side web controls. These controls are created and configured as objects, and automatically provide their own HTML output. Even better,ASP.NET allows web controls to behave like their Windows counterparts: maintaining state, and even raising events that you can react to in code.

This chapter explains the basics of the server controls. You'll learn how to handle events, interactwith control objects, and understand the object hierarchy of HTML server controls. These topics are entirely different from traditional ASP programming, and represent an innovative new way to program the Web.

A Simple Page Applet

We'll start our exploration with a simple example of how server-based controls work with a single- page applet. This type of program, which combines user input and the program output on the same page, is used to provide popular tools on many sites. Some examples include calculators formortgages, taxes, health or weight indices, and retirement savings plans, single-phrase translators, and stock tracking utilities.

Our example is a simple page that allows the user to convert a number of U.S. dollars to the equivalent amount of Euros (see Figure 6-1).

Figure 6-1: A simple currency converter

The HTML for this page is shown next. To make it as clear as possible, the style attribute of the tag used for the border has been omitted. There are two <input> tags: one for the text box,and one for the submit button. These elements are enclosed in a <form> tag, so they can be submitted back to the server when the button is clicked. The rest of the page is static text.

```
<HTML><body>
  <form method="post" >
```

```
  <div>

    Convert:  
    <input type="text">  US dollars to Euros.<br><br>
    <input type="submit" value="OK">

  </div>
 </form>
</body></HTML>
```

The Problem with Response.Write

In a traditional ASP program, you would add the currency conversion functionality to this page by examining the posted form values and manually writing the result to the end of the page with the Response.Write command. This approach works well for a simple page, but it encounters a number of difficulties as the program becomes more sophisticated:

"Spaghetti" code The order of the Response.Write statements determines the output. If you want to tailor different parts of the output based on a condition, you willneed to reevaluate that condition at several different places in your code.

Lack of flexibility Once you have perfected your output, it's very difficult to change it.If you decide to modify the page several months later, you have to read through the code, follow the logic, and try to sort out numerous details.

Confusing content and formatting Depending on the complexity of your user interface, your Response.Write may need to add new HTML tags and style attributes.This encourages programs to tangle formatting and content details together, making itdifficult to change just one or the other at a later date.

Complexity Your code becomes increasingly intricate and disorganized as you adddifferent types of functionality. For example, it could be extremely difficult to track theeffects of different conditions and different Response.Write blocks if you created a combined tax-mortgage-interest calculator.

Quite simply, an ASP.NET application that needs to create a sizable portion of interface using Response.Write commands encounters the same dilemmas that a Windows program would find ifit needed to manually draw its text boxes and command buttons on an application window in response to every user action.

Server Controls

In ASP.NET, you can still use Response.Write to create a dynamic web page. But ASP.NET provides a better approach. It allows you to turn static HTML tags into objects (called controls) thatyou can program on the server.

ASP.NET provides two sets of server controls:

- **HTML server controls** are server-based equivalents for standard HTML elements. These controls are ideal if you are a seasoned web programmer who prefers to work with familiarHTML tags (at least at first). They are also useful when migrating existing ASP pages to ASP.NET, as they require the fewest changes.

- **Web controls** are similar to the HTML server controls, but provide a richer object model witha variety of properties for style and formatting details, more events, and a closer parallel to Windows development. Web controls also feature some user interface elements that have no HTML equivalent, such as the DataGrid and validation controls. We examine web

controls in the next chapter.

HTML Server Controls

HTML server controls provide an object interface for standard HTML elements. They provide threekey features:

They generate their own interface. You set properties in code, and the underlying HTML tag is updated automatically when the page is rendered and sent to the client.

They retain their state. You can write your program the same way you would write a traditional Windows program. There's no need to recreate a web page from scratch each time you send it to the user.

They fire events. Your code can respond to these events, just like ordinary controls ina Windows application. In ASP code, everything is grouped into one block that executes from start to finish. With event-based programming, you can easily respond to individual user actions and create more structured code.

To convert the currency converter to an ASP.NET page that uses server controls, all you need todo is add the special attribute runat="server" to each tag that you want to transform into a server control. You should also add an id attribute to each control that you need to interact with in code.The id attribute assigns the unique name that you will use to refer to the control in code.

In the currency converter application, the input text box and submit button can be changed intoserver controls. In addition, the <form> element must also be processed as a server control to allow ASP.NET to access the controls it contains.

```
<HTML><body>
  <form method="post" runat="server">
    <div>

      Convert:  
      <input type="text" id="US" runat="server">  US dollars to
      Euros.
      <br><br>
      <input type="submit" value="OK" id="Convert" runat="server">

    </div>
  </form>
</body></HTML>
```

Viewstate

If you look at the HTML that is sent to you when you request the page, you'll see it is slightly different than the information in the .aspx file. First of all, the runat="server" attributes are strippedout (as they have no meaning to the client browser, which can't interpret them). More importantly,an additional hidden field has been added to the form.

```
<HTML><body>
  <form method="post" runat="server">
  <input type="hidden" name="_VIEWSTATE"
  value="dDw3NDg2NTI5MDg7Oz4=" />
    <div>

      Convert:  
      <input type="text" id="US" runat="server">  US dollars to
      Euros.
      <br><br>
      <input type="submit" value="OK" id="Convert" runat="server">
```

```
   </div>
  </form>
</body></HTML>
```

This hidden field stores information about the state of every control in the page in a compressedand lightly encrypted form. It allows you to manipulate control properties in code and have the changes automatically persisted. This is a key part of the web forms programming model, whichallows you to forget about the stateless nature of the Internet, and treat your page like a continuously running application.

ASP.NET Controls Maintain State Automatically

Even though the currency converter program does not yet include any code, you will already notice one change. If you enter information in the text box, and click the submit button to post thepage, the refreshed page will still contain the value you entered in the text box. (In the original example that uses ordinary HTML elements, the value will be cleared every time the page is posted back.)

The HTML Control Classes

Before you can continue any further with the currency calculator, you need to know about thecontrol objects you have created. All the HTML server controls are defined in the System.Web.UI.HtmlControls namespace. There is a separate class for each kind of control.Table 6-1 describes the basic HTML server controls.

So far, the currency converter defines three controls, which are instances of the HtmlForm, HtmlInputText, and HtmlInputButton classes, respectively. It's important that you know the class names, because you need to define each control class in the code-behind file if you want to interact with it. (Visual Studio .NET simplifies this task: whenever you add a control using its web designer, the appropriate tag is added to the .aspx file and the appropriate variables are defined inthe code-behind class.)

Table 6-1: The HTML Server Control Classes

Class Name	HTML Tag Represented
HtmlAnchor	<a>
HtmlButton	<button>
HtmlForm	<form>
HtmlImage	<img>
HtmlInputButton	<input type="button">, <input type="submit">, and <input type="reset">
HtmlInputCheckBox	<input type="checkbox">
HtmlInputControl	<input type="text">, <input type="submit">, and <input type="file">
HtmlInputFile	<input type="file">
HtmlInputHidden	<input type="hidden">
HtmlInputImage	<input type="image">
HtmlInputRadioButton	<input type="radio">
HtmlInputText	<input type="text"> and <input type="password">

HtmlSelect	<select>
HtmlTable, HtmlTableRow, and HtmlTableCell	<table>, <tr>, <th> and <td>
HtmlTextArea	<textarea>
HtmlGenericControl	Any other HTML element

The HTML Server Control Reference

This chapter introduces many new server controls. As we work our way through the examples, I'll explain many of the corresponding properties and events, and the class hierarchy. However, for single-stop shopping, you are encouraged to refer to Chapter 26, which provides a complete HTML server control reference. Each control is featured separately, with a picture, example tag, and a list of properties, events, and methods. In the meantime, Table 6-2 gives a quick overview of some important properties.

Table 6-2: Important Properties

Control	Most Important Properties
HtmlAnchor	Href, Target, Title
HtmlImage and HtmlInputImage	Src, Alt, Width, and Height
HtmlInputCheckBox and	CheckedHtmlInputRadioButton
HtmlInputText	Value
HtmlSelect	Items (collection)
HtmlTextArea	Value
HtmlGenericControl	InnerText

Improving the Currency Converter

Now that we've looked at the basics of server controls, it might seem that their benefits are fairly minor compared with the cost of learning a whole new system of web programming. In this next section, we'll start to extend the currency calculator utility. You'll see how additional functionality can be snapped in to our existing program in an elegant, modular way. As our program grows, theASP.NET framework handles its complexity easily, steering us away from the tangled and intricatecode that would be required in a traditional ASP application.

Adding Support for Multiple Currencies

First, we'll add a <select> element (named Currency) that allows the user to choose the destination currency. To reduce the amount of HTML programming, no actual <option> entries areadded to the drop-down list in the .aspx file.

```
<%@ Page Language="VB" Inherits="CurrencyConverter"
    Src="CurrencyConverter.vb"
    AutoEventWireup="False" %>

<HTML><body>
  <form method="post" runat="server">

    <!-- div tag used for formatting. -->
      Convert:  
      <input type="text" id="US" runat="server">  US dollars to
      Euros.
      <select style="WIDTH: 155px" id="Currency"
            runat="server"></select><br><br>
```

```
    <input type="submit" value="OK" id="Convert"
    runat="server"><br><br>
    <div style="FONT-WEIGHT: bold" id="Result" runat="server"></div>
   </div>

  </form>
</body></HTML>
```

Then we add the corresponding control variable to the CurrencyConverter class.

```
Protected Currency As HtmlSelect
```

The currency list is filled through code at runtime. In this case, the ideal event is the Page.Loadevent. This is the first event that occurs when the page is executed.

```
Private Sub Page_Load(sender As Object, e As EventArgs) _
  Handles MyBase.Load

   If Me.IsPostBack = False Currency.Items.Add("Euro")
      Currency.Items.Add("Japanese Yen")
      Currency.Items.Add("Canadian Dollars")
   End If

End Sub
```

Interesting Facts About this Code

Specify MyBase to handle a Page event. This keyword identifies the class that yourpage inherited from (in this case, the Page class).

Use the Items property with a list control. This allows you to append, insert, and remove <option> elements. Remember, when generating dynamic content with a server control, you set the properties, and the control creates the appropriate HTMLtags.

Check for postbacks. Before adding any items to this list, we need to make sure thatthis is the first time the page is being served. Otherwise, we will continuously add moreitems to the list or inadvertently overwrite the user's selection. The Me keyword points to the current instance of our page class. In other words, IsPostback is a property of the CurrencyConverter class, which CurrencyConverter inherited from the generic Page class.

Of course, if you are a veteran HTML coder, you know that a select list also provides a value attribute that you can use to store additional information. As our currency converter uses a shortlist of hard-coded values, this would be an ideal place to store the conversion rate.

To set the value tag, you need to create a ListItem object, and add that to the HtmlInputSelect control. The ListItem class provides a constructor that allows us to specify the text and value at thesame time that we create it, allowing condensed code like this:

```
Private Sub Page_Load(sender As Object, e As EventArgs) _
  Handles MyBase.Load

    If Me.IsPostBack = False
        ' The HtmlInputSelect control accepts text or ListItem objects.
        Currency.Items.Add(New ListItem("Euro", "1.12"))
        Currency.Items.Add(New ListItem("Japanese Yen", "122.33"))
        Currency.Items.Add(New ListItem("Canadian Dollar", "1.58"))
    End If

End Sub
```

To complete our example, the calculation code must be rewritten.

```
Private Sub Convert_ServerClick(sender As Object, e As EventArgs) _
  Handles convert.ServerClick
    Dim Amount As Decimal = Val(US.Value)

    ' Retrieve the select ListItem object by its index number.
    ' Each ListItem object provides a Text and Value property.Dim
    Item As ListItem
    Item = Currency.Items(Currency.SelectedIndex)

    Dim NewAmount As Decimal = Amount * Val(Item.Value)
    Result.InnerText = Amount.ToString() & " US dollars = "
    Result.InnerText &= NewAmount.ToString() & " " & Item.Text
End Sub
```

Figure 6-3 shows you what the converter will now look like.

Figure 6-3: The multi-currency converter

A Deeper Look at HTML Control Classes

Related classes in the .NET framework use inheritance to share functionality. For example, everyHTML control inherits from the base class HtmlControl, which provides some essential features every HTML server control uses. The inheritance diagram is shown in Figure 6-5.

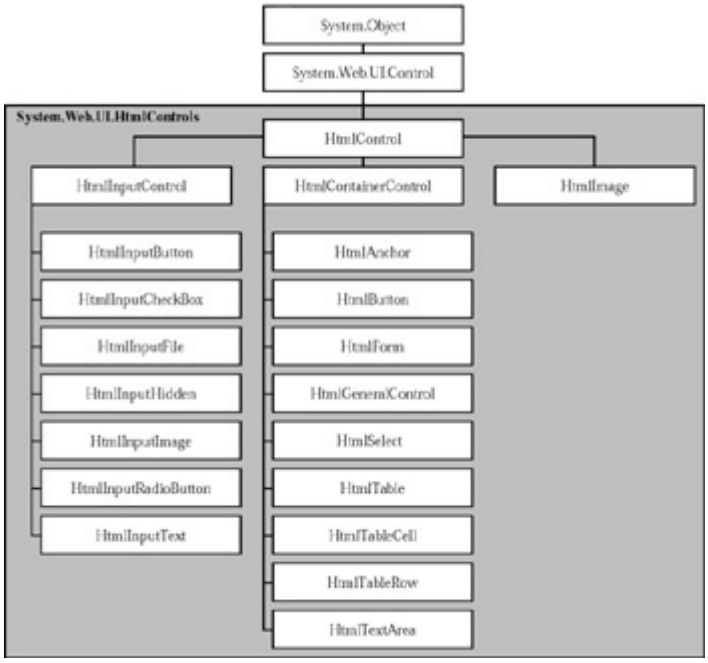


Figure 6-5: HTML control inheritance

The next few sections aim to shed some light on the ASP.NET classes used for controls. For thespecific details about each HTML control class, refer to Chapter 26.

HTML server controls generally provide properties that closely match their tag attributes (for example, the HtmlImage class provides Align, Alt, Border, Src, Height, and Width properties) andone of two possible events: ServerClick or ServerChange.

HTML Control Events

The ServerClick is simply a click that is processed on the server side. It's provided by most button

controls, and it allows your code to take immediate action. This action might override the expectedbehavior. For example, if you intercept the click event of an anchor control, the user won't be redirected to a new page unless you provide extra code to forward the request.

The ServerChange event responds when a change has been made to a text or selection control. This event is not as useful as it appears because it doesn't occur until the page is posted back (forexample, after the user clicks a submit button). At this point, the ServerChange event occurs for allchanged controls, followed by the appropriate ServerClick.

Table 6-3 shows which controls provide a ServerClick event, and which ones provide a ServerChange event.

Table 6-3: HTML Control Events

Event	Controls that Provide It
ServerClick	HtmlAnchor, HtmlForm, HtmlButton, HtmlInputButton, HtmlInputImage
ServerChange	HtmlInputText, HtmlInputCheckBox, HtmlInputRadioButton, HtmlInputHidden, HtmlSelect, HtmlTextArea

In Chapter 3, we introduced the standard for .NET events, which dictates that every event shouldpass exactly two pieces of information. The first parameter identifies the object (in this case, the control) that fired the event. The second parameter is a special object that can include additional information about the event.

Advanced Events with the HtmlInputImage Control

In the examples we've looked at so far, the second parameter (e) has always been used to passan empty System.EventArgs object. This object doesn't contain any additional information—it's just a glorified placeholder.

```
Private Sub Convert_ServerClick(sender As Object, e As EventArgs) _
  Handles convert.ServerClick
```

In fact, there is only one HTML server control that sends additional information: the HtmlInputImage control. It sends a special ImageClickEventArgs object (from the System.Web.UInamespace) that provides X and Y properties representing the location where the image was clicked.

```
Private Sub ImgButton_ServerClick(sender As Object, _
  e As ImageClickEventArgs) Handles ImgButton.ServerClick
```

This makes for an exceptionally easy way to replace multiple button controls and image maps when you need a sophisticated graphical user interface. The sample ImageTest.aspx page shownin Figure 6-6 puts this feature to work with a simple graphical button. Depending on whether the user clicks on the button border or on the button surface, a different message is displayed.

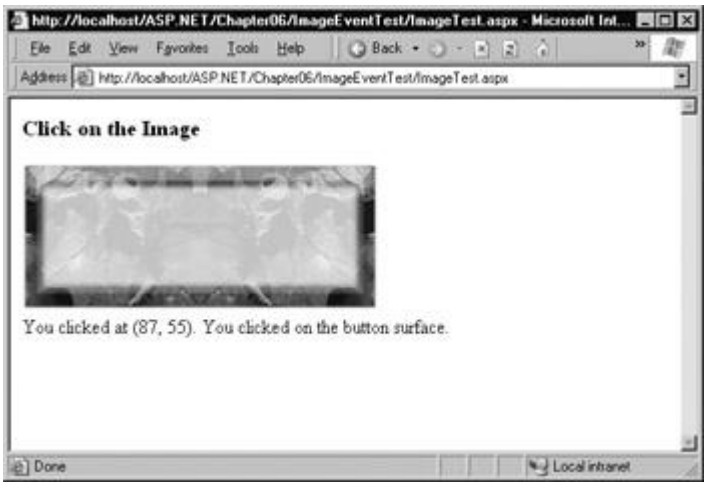


Figure 6-6: Using an HtmlInputImage control

The page code examines the click coordinates provided by the HtmllnputImage.ServerClick event.

```
Public Class ImageTest
  Inherits Page

    Protected Result As HtmlGenericControl
    Protected WithEvents ImgButton As HtmlInputImage

    Private Sub ImgButton_ServerClick(sender As Object, _
      e As ImageClickEventArgs) Handles ImgButton.ServerClick

        Result.InnerText = "You clicked at (" & e.X.ToString() & _", "
                    & e.Y.ToString() & "). "

        If e.Y < 100 And e.Y > 20 And e.X > 20 and e.X < 275 Then
            Result.InnerText &= "You clicked on the button surface."
        Else
            Result.InnerText &= "You clicked the button border."
        End If
    End Sub

End Class
```

The HtmlControl Base Class

Every HTML control inherits from the base class HtmlControl. This relationship means that everyHTML control will support a basic set of properties and features. These are shown in Table 6-4.

Table 6-4: HtmlControl Properties

Property	Description
Attributes	Provides a collection of all the tag attributes and their values. Ratherthan setting an attribute directly, it's better to use the corresponding property. However, this collection is useful if you need to add or configure a custom attribute or an attribute that doesn't have a corresponding property.
Controls	Provides a collection of all the controls contained inside the current control. (For example, a server control could contain an <input>server control.) Each object is provided as a generic System.Web.UI.Control object, so you may need to cast the referencewith the CType function to access control-specific properties.
Disabled	Set this to True to disable the control, ensuring that the user cannotinteract with it and its events will not be fired.

EnableViewState	Set this to False to disable the automatic state management for this control. In this case, the control will be reset to the properties and formatting specified in the control tag every time the page is posted back. If this is set to True (the default), the control uses the hidden input field to store information about its properties, ensuring that anychanges you make in code are remembered.
Page	Provides a reference to the web page that contains this control as a System.Web.UI.Page object.
Parent	Provides a reference to the control that contains this control. If the control is placed directly on the page (rather than inside another control), it will return a reference to the page object.
Style	Provides a collection of CSS style properties that can be used toformat the control.
TagName	Indicates the name of the underlying HTML element (for example,"img" or "div").
Visible	When set to False, the control will be hidden and will not be renderedto the final HTML page that is sent to the client.

The HtmlControl class also provides built-in support for data binding, which we examine in Chapter 14.

The HtmlContainerControl Class

Any HTML control that requires a closing tag also inherits from the HtmlContainer control. For example, elements such as <a>, <form>, and always use a closing tag, while <img> and <input> can be used as single tags. Thus, the HtmlAnchor, HtmlForm, and HtmlGenericControlclasses inherit from HtmlContainerControl, while HtmlImage and HtmlInput do not.

Properties Can Be Set in Code or in the Tag

To set the initial value of a property, you can configure the control in the Page.Load event handler,or you can adjust the control tag by adding special attributes. Note that the Page.Load event occurs after the page is initialized with the default values and the tag settings. That means that your code can override the properties set in the tag (but not vice versa).

The following HtmlImage control is an example that sets properties through attributes in the control tag. The control is automatically disabled and will not have its viewstate information stored.

```
<img EnableViewState="False" Disabled="True" id="Graph"
runat="server">
```

The only disadvantage with using control tag attributes is that errors will not necessarily be detected. An invalid property assignment in the code-behind, on the other hand, will almost alwaysgenerate a friendly runtime error informing you about the problem.

The HtmlContainerControl adds two properties, described in Table 6-5.

Table 6-5: HtmlContainerControl Properties

Property	Description
InnerHtml	The HTML content between the opening and closing tags of the control.

	Special characters that are set through this property will *not* be converted tothe equivalent HTML entities. This means you can use this property to applyformatting with nested tags like <b>, <i>, and <h1>.
InnerText	The text content between the opening and closing tags of the control. Special characters will be automatically converted to HTML entities and displayed like text (for example, the less than character (<) will be converted to < and will be displayed as < in the web page). That means that you can't use HTML tagsto apply additional formatting with this property. The simple currency converterpage used the InnerText property to enter results into a tag.

The HtmlInputControl Class

This control defines some properties (shown in Table 6-6) that are common to all the HTML controls based on the <input> tag, including the <input type="text">, <input type="submit">, and <input type="file"> elements.

Table 6-6: HtmlInputControl Properties

Property	Description
Type	Provides the type of input control. For example, a control based on <input type="file"> would return "file" for the type property.
Value	Returns the contents of the control as a string. In the simple currency converter, this property allowed the code to retrieve the information enteredin the text input control.

The Page Class

One control we haven't discussed in detail is the Page object. As explained in the previous chapter, every web page is a custom class that inherits from the System.Web.UI.Page control. By inheriting from this class, your page class acquires a number of properties that your code can use.These include properties for enabling caching, validation, and tracing, which are discussed in various chapters of this book.

Some of the more fundamental properties are described in Table 6-7, including the traditional built-in objects that you probably used in ASP programming, such as Response, Request, andSession.

Table 6-7: Basic Page Properties

Property	Description
Application and Session	These collections are used to hold state information on the server. Chapter 10 discusses this topic.
Cache	This collection allows you to store objects for reuse in other pages or for other clients. Caching is described in Chapter 23.
Controls	Provides a collection of all the controls contained on the webpage. You can also use the methods of this collection to addnew controls dynamically.
EnableViewState	When set to False, this overrides the EnableViewState property of the contained controls, ensuring that no controls will maintainstate information.
IsPostBack	This Boolean property indicates whether this is the first time thepage is being run (False), or whether the page is being resubmitted in response to a control event, typically with stored

	viewstate information (True). This property is often used in the Page.Load event handler, ensuring that basic setup is only performed once for controls that maintain viewstate.
Request	Refers to an HttpRequest object that contains information about the current web request, including client certificates, cookies, and values submitted through HTML form elements. Itsupports the same features as the built-in ASP Request object.
Response	Refers to an HttpResponse object that allows you to set theweb response or redirect the user to another web page. It supports the same features as the built-in ASP Response object, although it's used much less in .NET development.
Server	Refers to an HttpServerUtility object that allows you to perform some miscellaneous tasks, such as URL and HTML encoding. It supports the same features as the built-in ASP Server object.
User	If the user has been authenticated, this property will be initialized with user information. This property is described inmore detail when we examine security in Chapter 24.

The Controls Collection

The Page.Controls collection includes all the controls on the current web form. You can loop through this collection and access each control. For example, the following code writes out thename of every control on the current page to a server control called Result.

```
Result.InnerText = "List of controls: "
Dim ctrl As Control
For Each ctrl In Me.Controls
    Result.InnerText &= " " & ctrl.ID
Next
```

You can also use the Controls collection to add a dynamic control. The following code creates anew button with the caption "Dynamic Button" and adds it to the bottom of the page.

```
Dim ctrl As New HtmlButton()
ctrl.InnerText = "Dynamic Button"
Me.Controls.Add(ctrl)
```

This dynamically added button won't be stored in viewstate, so you will need to keep track of it andrecreate it after postbacks if you want it to remain.

The HttpRequest Class

The HttpRequest class encapsulates all the information related to a client request for a web page(see Table 6-8). Most of this information corresponds to low-level details such as posted-back form values, server variables, the response encoding, and so on. If you are using the ASP.NET framework to its fullest, you'll almost never dive down to that level. Other properties are generallyuseful for retrieving information, particularly about the capabilities of the client browser. Some of the security-related properties will return in our discussion in Chapter 24.

Table 6-8: HttpRequest Properties

Property	Description
ApplicationPath and PhysicalPath	ApplicationPath gets the ASP.NET application's virtual directory (URL), while PhysicalPath gets the "real" directory.
Browser	Provides a link to an HttpBrowserCapabilities object which contains

	properties describing various browser features, such as support for ActiveX controls, cookies, VBScript, and frames. This replaces the BrowserCapabilities component that was sometimes used in ASP development.
ClientCertificate	An HttpClientCertificate object that gets the security certificate forthe current request, if there is one.
Cookies	Gets the collection cookies sent with this request. Cookies are discussed in more detail in Chapter 10.
Headers and ServerVariables	Provides a name/value collection of HTTP headers and server variables. You can get the low-level information you need if you know the corresponding header or variable name.
IsAuthenticated and IsSecureConnection	Returns True if the user has been successfully authenticated and ifthe user is connected over secure sockets (also known as SSL or HTTPS).
QueryString	Provides the parameters that were passed along with the query string. Chapter 10 discusses how you can use the query string totransfer information between pages.
Url and UrlReferrer	Provides a Uri object that represents the current address for thepage, and the page where the user is coming from (the previouspage that linked to this page).
UserAgent	A string representing the browser type. Internet Explorer providesthe value "MSIE" for this property.
UserHostAddress and UserHostName	Gets the IP address and the DNS name of the remote client. Youcould also access this information through the ServerVariables collection.
UserLanguages	Provides a sorted string array that lists the client's language preferences. This can be useful if you need to create multilingualpages.

The HttpResponse Class

The HttpResponse class allows you to send information directly to the client. In traditional ASP development, the Response object was used heavily to create dynamic pages. Now, with the introduction of the new server-based control model, these relatively crude methods are no longerneeded.

The HttpResponse does still provide some important functionality: namely caching support, cookiefeatures, and the Redirect method that allows you to transfer the user to another page. A list of HttpResponse members is provided in Table 6-9.

```
' You can redirect to a file in the current directory.
Response.Redirect("newpage.aspx")

' You can redirect to another website.
Response.Redirect("http://www.prosetech.com")
```

Table 6-9: HttpResponse Members

Member	Description
BufferOutput	When set to True (the default), the page is not sent to the client until it is completely rendered and ready to send, rather than piecemeal.
Cache	References an HttpCachePolicy object that allows you to configure

	how this page will be cached. Caching is discussed in Chapter 23.
Cookies	The collection of cookies sent with the response. You can use this property to add additional cookies, as described in Chapter 10.
Write(), BinaryWrite(), and WriteFile()	These methods allow you to write text or binary content directly tothe response stream. You can even write the contents of a file. These methods are de-emphasized in ASP.NET, and shouldn't beused in conjunction with server controls.
Redirect()	This method transfers the user to another page in your applicationor a different web site.

The ServerUtility Class

The ServerUtility class provides some miscellaneous helper methods, as listed in Table 6-10. Themost commonly used are UrlEncode/UrlDecode and HtmlEncode/HtmlDecode. These functions change a string into a representation that can safely be used as part of a URL or displayed in a web page.

Table 6-10: ServerUtility Methods

Method	Description
CreateObject	Creates an instance of the COM object that is identified by its programmatic ID (progID). This is included for backward compatibility,as it will generally be easier to interact with COM objects using the .NET framework services.
HtmlEncode and HtmlDecode	Changes an ordinary string into a string with legal HTML charactersand back again.
UrlEncode and UrlDecode	Changes an ordinary string into a string with legal URL characters andback again.
MapPath	Returns the physical file path that corresponds to a specified virtual filepath on the web server.
Transfer	Transfers execution to another web page in the current application. This is similar to the Response.Redirect method, but slightly faster. Itcannot be used to transfer the user to a site on another web server.

For example, imagine you want to display this text on a web page:

Enter a word <here>

If you try to write this information to a page or place it inside a control, you may end up with thisinstead:

Enter a word

In this case, the browser has tried to interpret the text "<here>" as an HTML tag. A similar problemoccurs if you actually use valid HTML tags. For example, consider this text:

To bold text use the <b> tag.

Not only will the text "<b>" not appear, the browser will interpret it as an instruction to make the following text bold. To circumvent this automatic behavior, you need to convert potential problematic values to their special HTML equivalents. For example < becomes < in your finalHTML page, which the browser displays as the < character.

You can perform this transformation on your own, or you can circumvent the problem by using the InnerText property. When you set the contents of a control using InnerText, any illegal characters are automatically converted into their HTML equivalents. However, this won't help if you want to set a tag that contains a mix of embedded HTML tags and encoded characters. It also won't be ofany use for controls that don't provide an InnerText property, such as the Label web control we'll examine in the next chapter.

In these cases, you can use the HtmlEncode method to replace the special characters.

```
' Will output as "Enter a word &lt;here&gt;" in the HTML file, but the'
browser will display it as "Enter a word <here>".
ctrl.InnerHtml = Server.HtmlEncode("Enter a word <here>")
```

Or consider this example, which mingles real HTML tags with text that needs to be encoded.

```
ctrl.InnerHtml = "To <b>bold</b> text use the "
ctrl.InnerHtml &= Server.HtmlEncode("<b>") & " tag."
```

Figure 6-7 shows the results of successfully and incorrectly encoding special HTML characters. You can try out this sample as the HtmlEncodeTest.aspx page included with the examples for thischapter.

Figure 6-7: Encoding special HTML characters

The HtmlEncode method is particularly useful if you are retrieving values from a database, where you aren't sure if the text is HTML-legal. You can use the HtmlDecode method to revert the text toits normal form if you need to perform additional operations or comparisons with it in your code.

Similarly, the UrlEncode method changes text into a form that can be used in a URL. Generally,this allows information to work as a query string variable, even if it contains spaces and other characters that aren't URL-legal.

Assessing HTML Server Controls

HTML controls are a compromise between web controls and traditional ASP.NET programming. They use the familiar HTML elements, but provide a limited object-oriented interface. Essentially,HTML controls are designed to be straightforward, predictable, and automatically compatible withexisting programs. With HTML controls, the final HTML page that is sent to the client closely resembles the original .aspx page.

In the next chapter, we introduce web controls, which provide a more sophisticated object interface that abstracts away the underlying HTML. If you are starting a new project, or need toadd some of ASP.NET's most powerful controls, web controls are the best option.

Web Controls

Overview

The last chapter introduced one of ASP.NET's most surprising revolutions: the change to event-driven and control-based programming. This change allows you to create programs for the Internet following the same structured, modern style you would use for a Windows application.

However, HTML server controls really only show a glimpse of what is possible with ASP.NET's new control model. To see some of the real advantages provided by this shift, you need to exploreweb controls, which provide a rich, extensible set of control objects. In this chapter, we'll examine the basic web controls and their class hierarchy. We'll also delve deeper into ASP.NET's event handling, and examine the web page lifecycle.

Stepping Up to Web Controls

Now that you've seen the new model of server controls, you might wonder why we need additionalweb controls. But in fact, HTML controls are still more limited than server controls need to be. For example, every HTML control corresponds directly to an HTML tag, meaning that you are bound by the limitations and abilities of the HTML language. Web controls, on the other hand, emphasizethe future of web design:

They provide rich user interface. A web control is programmed as an object, but doesn't necessarily correspond to a single tag in the final HTML page. For example, you might create a single Calendar or DataGrid control, which will be rendered as dozens of HTML elements in the final page. When using ASP.NET programs, you don'tneed to know anything about HTML. The control creates the required HTML tags for you.

They provide a consistent object model. The HTML language is full of quirks and idiosyncrasies. For example, a simple text box can appear as one of three elements,including <textarea>, <input type="text">, and <input type="password">. With web controls, these three elements are consolidated as a single TextBox control.
Depending on the properties you set, the underlying HTML element that ASP.NET renders may differ. Similarly, the names of properties don't follow the HTML attributenames. Controls that display text, whether it is a caption or a user-editable text box, expose a Text property.

They tailor their output automatically. ASP.NET server controls can detect the typeof browser, and automatically adjust the HTML code they write to take advantage of features such as support for DHTML. You don't need to know about the client becauseASP.NET handles that layer and automatically uses the best possible set of features.

They provide high-level features. You'll see that web controls allow you to access additional events, properties, and methods that don't correspond directly to typical HTML controls. ASP.NET implements these features by using a combination of tricks.

The Examples in this Book Use Web Controls

Throughout this book, I'll show examples that use the full set of web controls. In order to master ASP.NET development, you need to become comfortable with these user interface ingredients and understand all their abilities. HTML server controls, on the other hand, are less important for web development, unless you need to have a fine-grained control over the HTML code that will be

generated and sent to the client. They are de-emphasized.

Basic Web Control Classes

If you have created Windows applications before, you are probably familiar with the basic set of standard controls, including labels, buttons, and text boxes. ASP.NET provides web controls for allthese standbys. (If you've created .NET Windows applications, you'll notice that the class names and properties have many striking similarities, which are designed to make it easy to transfer the experience you acquire in one framework to another.)

Table 7-1 lists the basic control classes, and the HTML elements they generate. Note that somecontrols, such as Button and TextBox, can be rendered as different HTML elements. ASP.NET uses the element that matches the properties you have set. Also, some controls have no single HTML equivalent. For example, the CheckBoxList and RadioButtonList controls output as a <table> that contains multiple HTML checkboxes or radio buttons. ASP.NET wraps them into asingle object on the server side for convenient programming, illustrating one of the primary strengths of web controls.

Table 7-1: Basic Web Controls

Control Class	Underlying HTML Element
Label	
Button	<input type="submit"> or <input type="button">
TextBox	<input type="text">, <input type="password">, or <textarea>
CheckBox	<input type="checkbox">
RadioButton	<input type="radio">
Hyperlink	<a>
LinkButton	<a> with a contained <img> tag
ImageButton	<input type="image">
Image	<img>
ListBox DropDownList	<select size="X"> where X is the number of rows that are visible atonce <select>
CheckBoxList	A list or <table> with multiple <input type="checkbox"> tags
RadioButtonList	A list or <table> with multiple <input type="radio"> tags Panel
Table, TableRow, and	<table>, <tr>, and <td> or <th>
TableCell	

This table omits some of the more specialized rich controls, which we'll see in Chapter 9, and the advanced data controls, which we'll see in Chapter 15.

AutoPostBack and Web Control Events

The previous chapter explained that one of the main limitations of HTML server controls is their

limited set of useful events—exactly two. HTML controls that trigger a postback, such as buttons,raise a ServerClick event. Input controls provide a ServerChange event that won't be detected until the page is posted back.

Server controls are really an ingenious illusion. You'll recall that the code in an ASP.NET page is processed on the server. It's then sent to the user as ordinary HTML (see Figure 7-7).

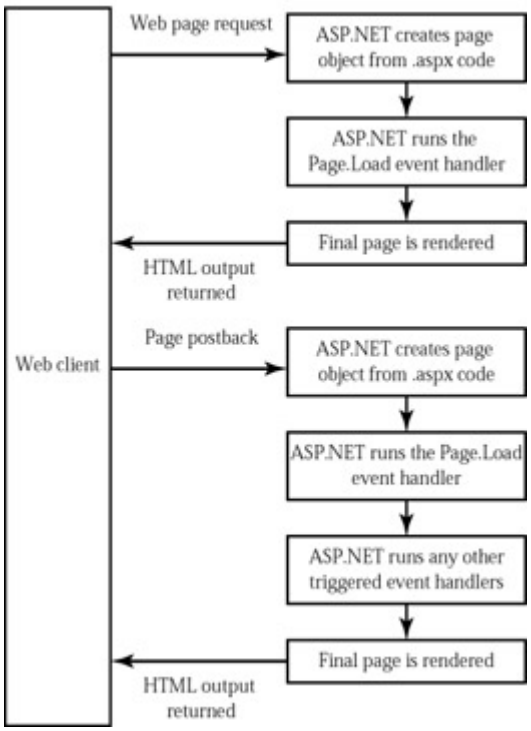


Figure 7-7: The page processing sequence

This is the same in ASP.NET as it was in traditional ASP programming. The question is, how canyou write server code that will react to an event that occurs on the client? The answer is a new innovation called the automatic postback.

Automatic postback submits a page back to the server when it detects a specific user action. Thisgives your code the chance to run again and create a new, updated page. Controls that support automatic postbacks include almost all input controls. A basic list of web controls and their eventsis provided in Table 7-4.

Table 7-4: Web Control Events

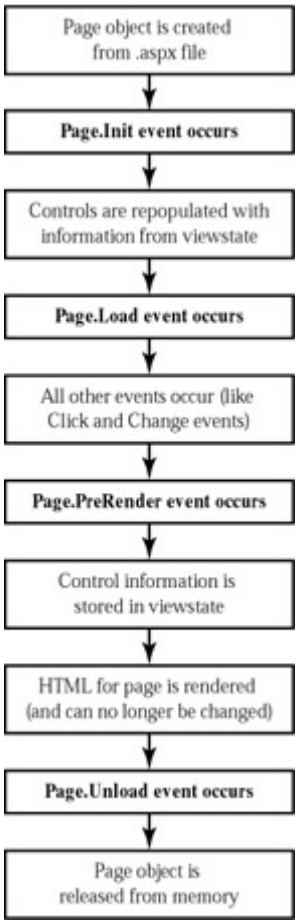
Event	Web Controls that Provide It
Click	Button, ImageButton
TextChange	TextBox (only fires after the user changes the focus to another control)
CheckChanged	CheckBox, RadioButton
SelectedIndexChanged	DropDownList, ListBox, CheckBoxList, RadioButtonList

If you want to capture a change event for a web control, you need to set its AutoPostBack propertyto True. That means that when the user clicks a radio button or checkbox, the page will be resubmitted to the server. The server examines the page, loads all the current information, and then allows your code to perform some extra processing before returning the page back to the user.

In other words, every time you need to update the web page, it is actually being sent to the server

and recreated (see Figure 7-8). However, ASP.NET makes this process so transparent that yourcode can treat your web page like a continuously running program that fires events.

Figure 7-8: The postback processing sequence



This postback system is not ideal for all events. For example, some events that you may be familiar with from Windows programs, such as mouse movement events or key press events, are not practical in an ASP.NET application. Resubmitting the page every time a key is pressed or themouse is moved would make the application unbearably slow and unresponsive.

How Postback Events Work

The AutoPostBack feature uses a couple of interesting tricks and a well-placed JavaScript functionnamed doPostBack.

```
<script language="javascript">
<!--
   function___doPostBack(eventTarget, eventArgument) {var
      theform = document.Form1;
      theform._EVENTTARGET.value = eventTarget; theform.
      EVENTARGUMENT.value = eventArgument;
      theform.submit();
   }
// -->
</script>
```

ASP.NET adds the JavaScript code to the page automatically if any of your controls are set to use AutoPostBack. It also adds two additional hidden input fields that are used to pass information back to the server: namely, the ID of the control that raised the event, and any additional information that might be relevant.

```
<input type="hidden" name="__EVENTTARGET" value="" />

<input type="hidden" name="__EVENTARGUMENT" value="" />
```

Finally, the control that has its AutoPostBack property set to True is connected to the __doPostBack function using the onclick or onchange attribute. The following example shows alist control called lstBackColor, which posts back automatically by calling the __doPostBack function.

```
<select id="lstBackColor" onchange="__doPostBack('lstBackColor','')"
 language="javascript">
```

In other words, ASP.NET automatically changes a client-side JavaScript event into a server-side ASP.NET event, using the __doPostBack function as an intermediary. It's possible that you may have created a solution like this manually for traditional ASP programs. The ASP.NET frameworkhandles these details for you automatically, simplifying life a great deal.

A Simple Web Page Applet

Now that you've had a whirlwind tour of the basic web control model, it's time to put it to work withour second single-page utility. In this case, it's a simple example for a dynamic e-card generator (see Figure 7-10). You could extend this sample (for example, allowing users to store e-cards to adatabase, or using the techniques in Chapter 16 to mail notification to card recipients), but on its own it represents a good example of basic control manipulation.

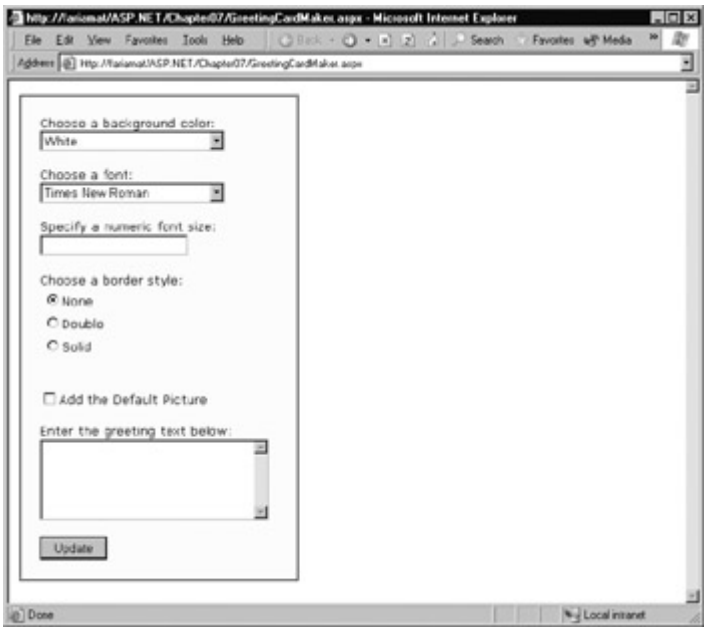


Figure 7-10: The card generator

The page is divided into two regions. On the left is an ordinary tag containing a set of webcontrols for specifying card options. On the right is a Panel control (named pnlCard), which

contains two other controls (lblGreeting and imgDefault) that are used to display user-configurabletext and a picture.

Whenever the user clicks the OK button, the page is posted back, and the "card" is updated (seeFigure 7-11).

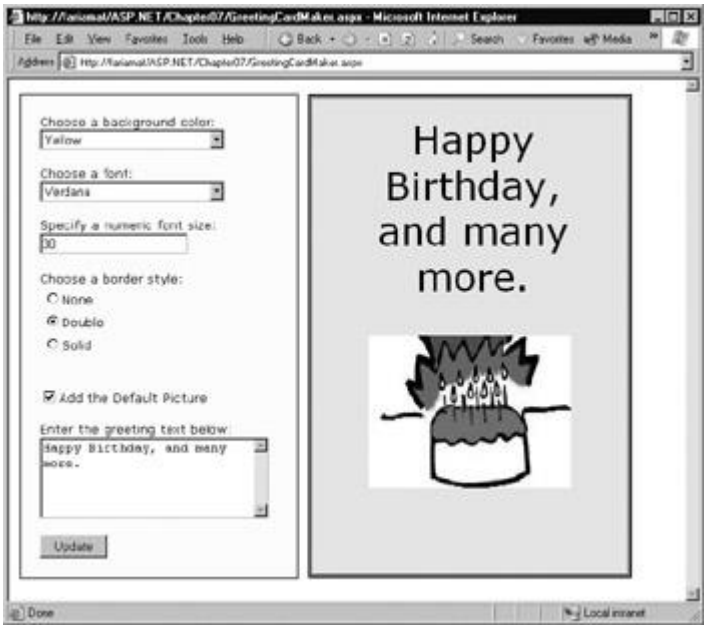


Figure 7-11: A user-configured greeting card

The .aspx layout code is very straightforward. Of course, the sheer length of it makes it difficult towork with efficiently. This is an ideal point to start considering Visual Studio .NET, which will handle the .aspx details for you automatically, and not require you to painstakingly format and organize the user interface markup tags.

```
<%@ Page Language="VB" Src="GreetingCardMaker.vb"
    Inherits="GreetingCardMaker"
    AutoEventWireup="False"%>

<HTML><body>
  <form method="post" runat="server">
  <!-- div formatting tag left out for clarity. -->

  <!-- Here are the controls: -->
  Choose a background color:<br>
  <asp:DropDownList id="lstBackColor" runat="server" Width="194px"
      Height="22px"/><br><br>
  Choose a font:<br>
  <asp:DropDownList id="lstFontName" runat="server" Width="194px"
      Height="22px" /><br><br>
  Specify a numeric font size:<br>
  <asp:TextBox id="txtFontSize" runat="server" /><br><br>
  Choose a border style:<br>
  <asp:RadioButtonList id="lstBorder" runat="server" Width="177px"
      Height="59px" /><br><br>
  <asp:CheckBox id="chkPicture" runat="server"
      Text="Add the Default Picture"></asp:CheckBox><br><br>
  Enter the greeting text below:<br>
  <asp:TextBox id="txtGreeting" runat="server" Width="240px"
      Height="85px"
      TextMode="MultiLine" /><br><br>
  <asp:Button id="cmdUpdate" runat="server" Width="71px"
      Height="24px"
      Text="Update" />
  </div>
```

```
<!-- Here is the card: -->
<asp:Panel id="pnlCard" style="Z-INDEX: 101; LEFT: 313px; POSITION:
    absolute;
    TOP: 16px" runat="server" Width="339px" Height="481px"
    HorizontalAlign="Center"><br> 
<asp:Label id="lblGreeting" runat="server" Width="256px"
    Height="150px" /><br><br><br>
<asp:Image id="imgDefault" runat="server" Width="212px"
    Height="160px" />
</asp:Panel>

</form>
</body></HTML>
```

The code follows the familiar pattern with an emphasis on two events: the Page.Load event, where initial values are set, and the Button.Click event, where the card is generated. I've left out the Imports statements in the following listing, as the basic set of required namespaces should befamiliar by now.

```
Public Class GreetingCardMaker
  Inherits Page
    Protected lstBackColor As DropDownList
    Protected lstFontName As DropDownList
    Protected txtFontSize As TextBox Protected
    chkPicture As CheckBox Protected
    txtGreeting As TextBox Protected pnlCard
    As Panel
    Protected lblGreeting As Label Protected
    lstBorder As RadioButtonListProtected
    imgDefault As Image
    Protected WithEvents cmdUpdate As Button

    Private Sub Page_Load(sender As Object, e As EventArgs) _
      Handles MyBase.Load
        If Me.IsPostBack = False Then

            ' Set color options. lstBackColor.Items.Add("White")
            lstBackColor.Items.Add("Red")
            lstBackColor.Items.Add("Green")
            lstBackColor.Items.Add("Blue")
            lstBackColor.Items.Add("Yellow")

            ' Set font options. lstFontName.Items.Add("Times
            New Roman")lstFontName.Items.Add("Arial")
            lstFontName.Items.Add("Verdana")
            lstFontName.Items.Add("Tahoma")

            ' Set border style options by adding a series of'
            ListItem objects.
            ' Each item indicates the name of the option,' and
            contains the corresponding integer
            ' in the Value property.
            lstBorder.Items.Add(New _
               ListItem(BorderStyle.None.ToString(), BorderStyle.None))
            lstBorder.Items.Add(New _
                ListItem(BorderStyle.Double.ToString(), _
                BorderStyle.Double))
            lstBorder.Items.Add(New _
                ListItem(BorderStyle.Solid.ToString, _
                BorderStyle.Solid))

            ' Select the first border option.
            lstBorder.SelectedIndex = 0

            ' Set the picture.
```

```
        imgDefault.ImageUrl = "defaultpic.png"
    End If
End Sub

Private Sub cmdUpdate_Click(sender As Object, e As EventArgs) _
  Handles cmdUpdate.Click

    ' Update the color.
    pnlCard.BackColor = Color.FromName( _
                lstBackColor.SelectedItem.Text)

    ' Update the font.
    lblGreeting.Font.Name = lstFontName.SelectedItem.Text

    If Val(txtFontSize.Text) > 0 Then
    lblGreeting.Font.Size = FontUnit.Point( _
                    Val(txtFontSize.Text))
    End If

    ' Update the border style.
    pnlCard.BorderStyle = Val(lstBorder.SelectedItem.Value)

    ' Update the picture.
    If chkPicture.Checked = True Then
        imgDefault.Visible = True
    Else
        imgDefault.Visible = False
    End If

    ' Set the text.
    lblGreeting.Text = txtGreeting.Text

End Sub

End Class
```

As you can see, this example limits the user to a few preset font and color choices. The code for the BorderStyle option is particularly interesting. The lstBorder control has a list that displays the text name of one of the BoderStyle enumerated values. You'll remember from the introductory chapters that every enumerated value is really an integer with a name assigned to it. The lstBorder also secretly stores the corresponding number, so that it can set the enumeration easilyin the cmdUpdate_Click event handler.

Validation and Rich Controls

Overview

The ASP.NET web control framework has almost unlimited possibilities. In the coming months, we'll see third-party component developers create increasingly sophisticated control classes that you can plug in to your web applications effortlessly. In this chapter, we start to look at some of thereal promise of ASP.NET and the server-control model, and consider controls that have no equivalent in the ordinary HTML world: the Calendar and AdRotator.

The second part of this chapter examines ASP.NET's validation controls. These controls take a previously time-consuming and complicated task—verifying user input and reporting errors—and automate it with an elegant, easy-to-use collection of validators. You'll learn how to add these controls to an existing page, and use regular expression, custom validation functions, and manualvalidation. And as usual, we'll peer under the hood to take a look at how ASP.NET implements these new features.

Finally, we'll briefly consider the next generation of controls, which promises to bring a new worldof rich user interface to the Web. These new controls, prepared by other developers, represent one of the most exciting new directions for ASP.NET.

The Calendar Control

The Calendar control is one of the most impressive web controls. It's commonly called a rich control because it can be programmed as a single object (and defined in a single, simple tag), butrendered in dozens of lines of HTML output.

```
<asp:Calendar id="Dates" runat="server" />
```

In its default mode, the Calendar control presents a single month view (see Figure 9-1) where theuser can navigate from month to month using the navigational areas, at which point the page is posted back, and ASP.NET automatically provides a new page with the correct month values.
When the user clicks on a date, it becomes highlighted in a gray box. You can retrieve the day asa DateTime object from the Calendar.SelectedDate property.

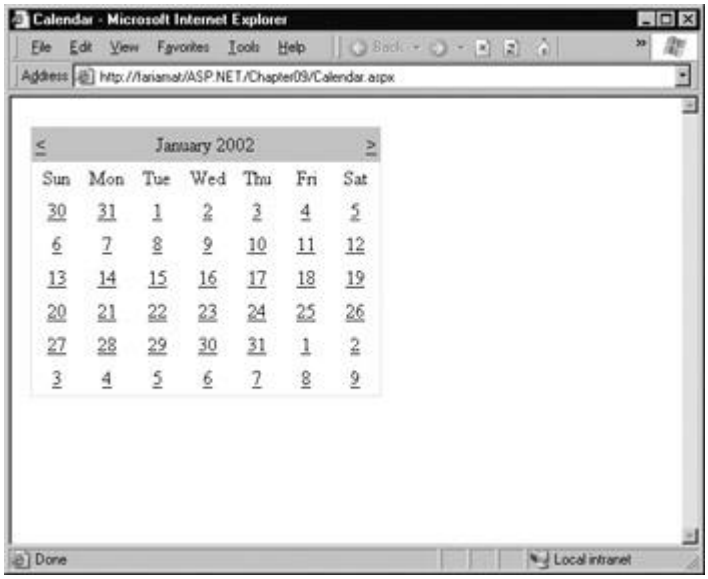


Figure 9-1: The default Calendar

This basic set of features may provide everything you need in your application. Alternatively, you can configure different selection modes to allow users to select entire weeks or months, or render

the control as a static calendar that doesn't allow selection. The important fact to remember is thatif you allow month selection, the user can also select a single week or a day. Similarly, if you allowweek selection, the user can also select a single day.

The type of selection is set through the Calendar.SelectionMode property. You may also need to set the Calendar.FirstDayOfWeek property to configure how a week is selected. (For example, setFirstDayOfWeek to the enumerated value Monday, and weeks will be selected from Monday to Sunday.)

When you allow multiple date selection, you need to examine the SelectedDates property, whichprovides a collection of all the selected dates. You can loop through this collection using the For Each syntax. The following code demonstrates this technique (see Figure 9-2).

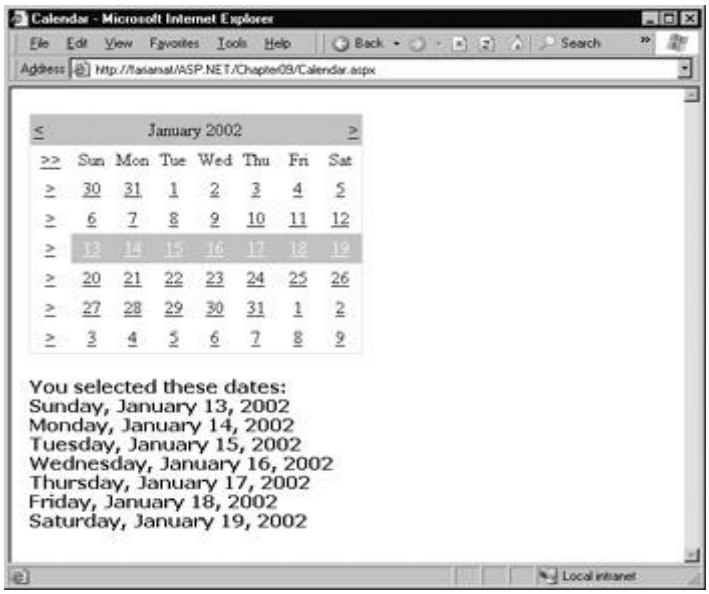


Figure 9-2: Selecting multiple dates

```
lblDates.Text = "You selected these dates:<br>"Dim

dt As DateTime
For Each dt In MyCalendar.SelectedDates lblDates.Text
    &= dt.ToLongDateString() & "<br>"
Next
```

Formatting the Calendar

The Calendar control provides a whole host of formatting-related properties. These are described in more detail in Chapter 27. It's enough to note that various parts of the Calendar, like the header,selector, and various day types can be set using one of the style properties (for example, WeekendDayStyle). Each of these style properties references a full-featured TableItemStyle object that provides properties for coloring, border style, font, and alignment. Taken together, they allow you to modify almost any part of the Calendar's appearance.

If you are using an IDE such as Visual Studio .NET, you can even set an entire related color scheme using the built-in designer. Simply right-click on the control on your design page, andselect Auto Format. You will be presented with a list of predefined formats that set the style properties, as shown in Figure 9-3.

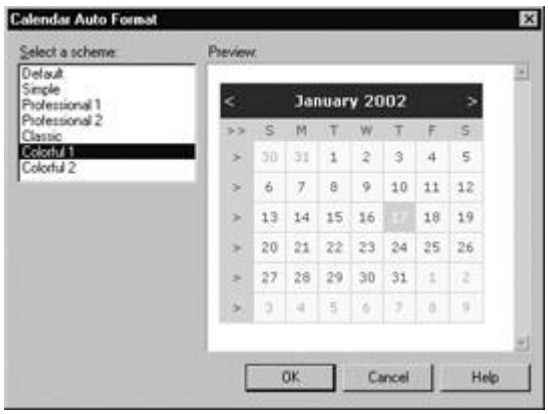


Figure 9-3: Calendar styles

You can also use additional properties to hide some elements or configure the text they display.

Restricting Dates

In most situations where you need to use a calendar for selection, you don't want to allow the userto select any date in the calendar. For example, the user might be booking an appointment or choosing a delivery date, two services that are generally only provided on set days. The Calendarmakes it surprisingly easy to implement this logic. In fact, if you've worked with the date and time controls on the Windows platform, you'll quickly recognize that the ASP.NET versions are far superior.

The basic approach to restricting dates is to write an event handler for the Calendar.DayRender event. This event occurs when the Calendar is about to create a month to display to the user. Thisevent gives you the chance to examine the date that is being added to the current month (throughthe e.Day property), and decide whether it should be selectable or restricted.

```
Private Sub DayRender(source As Object, e As DayRenderEventArgs) _Handles
  Calendar.DayRender

    ' Restrict dates after the year 2100, and those on the weekend.
    If e.Day.IsWeekend Or e.Day.Date.Year > 2100 Then
        e.Day.IsSelectable = False
    End If

End Sub
```

The e.Day object is an instance of the CalendarDay class, which provides various useful properties. These are described in Table 9-1.

Table 9-1: CalendarDay Properties

Property	Description
Date	The DateTime object that represents this date.True
IsWeekend	if this date falls on a Saturday or Sunday.
IsToday	True if this value matches the Calendar.TodaysDate property, which isset to the current day by default.
IsOtherMonth	True if this date does not belong to the current month, but is displayed tofill in the first or last row. For example, this might be the last day of the previous month or the next day of the following month.
IsSelectable	Allows you to configure whether the user can select this day.

The DayRender event is extremely powerful. Besides allowing you to tailor what dates are selectable, it also allows you to configure the cell where the date is located through the e.Cell

property (the Calendar is really a sophisticated HTML table). For example, you could highlight an important date or even add extra information (see Figure 9-4).

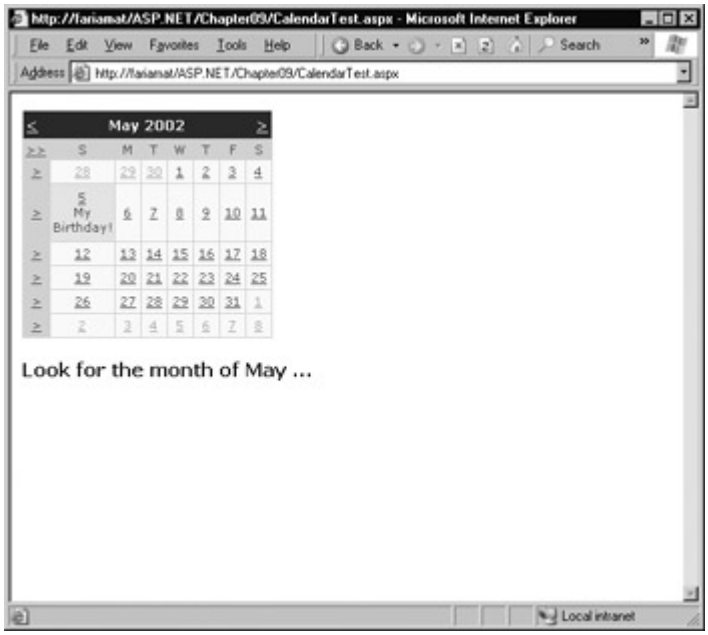


Figure 9-4: Highlighting a day

```
Private Sub DayRender(source As Object, e As DayRenderEventArgs) _Handles
  Calendar.DayRender

    ' Check for May 5 in any year, and format it.
    If e.Day.Date.Day = 5 And e.Day.Date.Month = 5 Then
        e.Cell.BackColor = System.Drawing.Color.Yellow

        ' Add some static text to the cell.
        Dim lbl As New Label lbl.Text
        = "<br>My Birthday!"
        e.Cell.Controls.Add(lbl)
    End If

End Sub
```

The Calendar control provides two other useful events: SelectionChanged and VisibleMonthChanged. These occur immediately after a change, but before the page is returned tothe user. You can react to this event and update other portions of the web page to correspond to the current calendar month (for example, setting a list of valid times in a list control).

```
Private Sub SelectionChanged(source As Object, e As EventArgs) _
  Handles Calendar.SelectionChanged

    lstTimes.Items.Clear

    Select Case Calendar.SelectedDate.DayOfWeekCase
        DayOfWeek.Monday
            ' Apply special Monday schedule.
            lstTimes.Items.Add("10:00")
            lstTimes.Items.Add("10:30")
            lstTimes.Items.Add("11:00")
        Case Else
            lstTimes.Items.Add("10:00")
            lstTimes.Items.Add("10:30")
            lstTimes.Items.Add("11:00")
            lstTimes.Items.Add("11:30")
            lstTimes.Items.Add("12:00")
            lstTimes.Items.Add("12:30")
    End Select
End Sub
```