




CHAPTER I

Configuring Dreamweaver for ASP.NET

- Defining a site
 - Setting up your IIS environment for production
 - Configuring Dreamweaver MX for use in versioning software
 - Managing the Web.config file
- 

Before you do anything with Dreamweaver MX, you'll need to figure out how it works at the site level. In other words, how can you configure it to build and deploy your website? Besides the normal configuration issues you'll encounter when working with Dreamweaver MX, such as the site definitions that all users need to learn, you'll want to get familiar with some of the intricacies of .NET and how they relate to decisions you make as your projects get underway.

.NET serves pages by compiling them at runtime into an intermediary language. Sometimes, some or all of the compilation takes place before runtime, such as when you develop what is called a *code-behind page*, but I'll save the intricacies of that for Chapter 9, "Advanced Coding with Dreamweaver MX." This intermediary language I'm talking about is basically machine code that is generated by ASP.NET from any number of sources written in languages such as C#, VB.NET, or even JavaScript. This is why you find so many different ways to work with .NET. Whatever language is being used, the pages ultimately get converted into machine code that tells the application server how to render your HTML.

All your code resides in either the code-behind page I just mentioned, or, more frequently for the purposes of this book because of the way Dreamweaver outputs ASP.NET pages, within special script tags. These tags have a `runat="server"` attribute value pair that is read exclusively by the server. The server interprets the script between the script tags and processes any functions you've defined within those tags. By including `runat="server"` in the script element, you prevent the contents of that tag from being interpreted by the browser, because ASP.NET compiles the page and returns HTML based on the instructions on the ASP.NET page you created.

One of the chief advantages of .NET is this compilation scheme, because it allows you or your loved ones to create custom controls in your favorite language and compile them as DLLs. Then, when you slip one of these DLLs into the `bin` directory of your .NET-based web application, .NET can use it to generate pages according to the functions it contains.

This is where Dreamweaver MX comes in. The good folks at Macromedia have already written a killer custom control for you, and it's where all the magic happens. With Dreamweaver MX, you can simply choose commands from the Dreamweaver interface to generate database-driven code. Dreamweaver generates special ASP.NET tags that tell ASP.NET how to interact with the Dreamweaver custom control. The amount of hand-coding time you can save is almost astonishing, and you'll save even more when you start your application development efforts with proper configuration settings. This chapter explores all these configuration issues, to get you off to the best possible start.

Defining a Site

Your first activity when you launch Dreamweaver MX, unless you're just looking at it to gain familiarity, will probably involve setting up a site using the Site menu to define your website. This will include telling Dreamweaver MX what language you want to use to create your .NET applications, and the specifics about accessing your site. This is a necessary step to take, since you need access to a server that can process .NET applications.

There are two ways to set up a site in Dreamweaver: Use a wizard, or choose the Advanced tab of the Site Definition panel. When the program is first launched after installation, it defaults to the wizard. However, if you choose the Advanced tab to define your site, the next time you start a new site definition the program will default to Advanced tab setup. To change it back to the wizard, click the Basic tab at the top of the Site Definition panel.

Using the Site Definition Wizard

Start off by choosing Site > Manage Sites. A dialog box labeled Manage Sites will appear. Click the New... button. When you do this, you'll see two options appear: Site, and FTP & RDS Server. Choose Site. A new, larger dialog box named Site Definition will appear. Click the tab labeled Basic. You'll see a Site Definition dialog box that looks similar to that in Figure 1.1.

FIGURE 1.1:

The Site Definition wizard



Here are the steps for completing the wizard:

1. The wizard asks you what you'd like to name your site. Give the site a descriptive name, and click the Next button.
2. The wizard next asks you if you want to work with a server technology such as ColdFusion, ASP.NET, ASP, JSP, or PHP. Under the question are two radio buttons. Choose the second one, "Yes, I want to use a server technology." When you click that, another question appears along with a drop-down list of choices. In this book we're using ASP.NET C#, so choose that from the list.
3. The next page asks you, "How do you want to work with your files during development?" Four radio buttons below that question reveal the following choices:

Edit and test locally (my testing server is on this computer) This means you want to edit and test from the same directory on your machine. If your computer has IIS (Microsoft's Web Server, Internet Information Services), Dreamweaver MX should know about it and will display a message indicating so at the bottom of the wizard. "Edit and test locally" is the default selection because most people like to test and develop locally, then deploy to the production site. In this dialog box you will need to choose where to store your files. Enter that information into the text field "Where on your computer do you want to store your files?" The wizard will suggest a folder for you, but you can override that choice with your own.

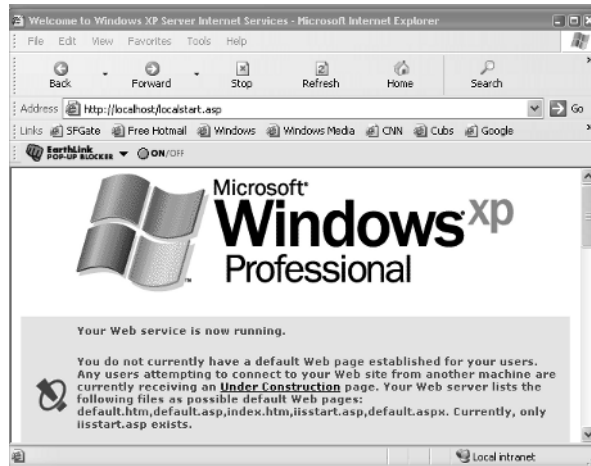
Edit locally, then upload to remote testing server This means you want to edit local files on your hard drive, then upload them to a server for testing. The testing server is not your production server. Enter the location for storing your files into the text field "Where on your computer do you want to store your files?" You can accept the default choice or select your own folder.

Edit directly on remote testing server using local network This means you want to set up a directory on your local machine that also serves as the root source for a local testing server using HTTP. You'll need to have a web server configured in order for this to work because the wizard will ask you for a testing server URL. This is usually `http://localhost` but can also be `http://127.0.0.1` (which is the same thing as `localhost`).

To find out if your server is running, type either of these locations into your web browser. If the server is running and you haven't performed any custom configurations, your browser will generally display a page like that shown in Figure 1.2. Once you're certain your server is running, fill in the text field "Where are your files on the network?" You'll also choose this if you are in a workgroup working on a networked server. Either way, access is provided through a LAN, and you need to have access to a live server.

FIGURE 1.2:

When you type in `http://localhost`, your browser will show an IIS page similar to this.



Edit directly on remote testing server using FTP or RDS Clicking this option reveals a bit more information about your editing choice, informing you that Dreamweaver will need to download files from an FTP site. So the wizard asks you, “Where on your computer do you want Dreamweaver to store local copies of your files?” You can browse to the location or input the path directly into the text field. This is where Dreamweaver will store local copies of files downloaded from the FTP site. It’s a reasonable choice if you are working with a remote hosting service and you’re worried that testing on your own server won’t be a true gauge of your site’s performance.

TIP

You can set up a sandbox on your remote site by creating a directory within the root of your domain and calling it something like “test_root”. Then you can mirror your site and copy everything onto your production server exactly as mapped out in your test directory.

4. The next wizard page will depend on your choice from Step 3. Let’s do an “if ... then” routine to map out your options:

Edit and test locally If you chose this option, the next page contains a dialog box with a text field “What URL would you use to browse to the root of your site?” An example of what you might enter in this field might be something like `http://localhost/mySite/`.

Edit locally, then upload to remote testing server If you chose this option, you’ll be presented with a screen with a drop-down list at the top, labeled “How do you connect to your testing server?” The four options available are “I’ll set this up later,” FTP, Local/Network, and RDS. Forget about RDS since that’s specific to ColdFusion.

- You can choose “I’ll set this up later,” and move to the next page.

- If you select FTP, the dialog box changes and presents options for filling in your FTP connection information. The hostname or FTP address of your server should be the same as your website, unless your host provider or system administrator has given you alternate FTP information. Whether you need to fill in the text field “What folder on the testing server do you want to store your files in?” depends on whether the site is at the root of the FTP address you’ve been assigned. If you don’t know, contact your host provider or system administrator to be sure. You’ll also need your username and password in order to successfully connect to your FTP service. These, too, are provided by your host provider or network administrator.
- If you choose Local/Network, choose a folder the test files should be stored in in the text field “Where do you copy your files to in order to test them?” You can click the small folder icon next to the text field to browse to the folder that should store your test files.

Edit directly on remote testing server using local network If you chose this option, the next wizard page asks, “What URL would you use to browse to the root of the site?” The answer to this will vary, and it’s important to get it right, so if you’re not administering your own site, be sure to check with your system administrator to be sure you enter the correct information. Dreamweaver manages links based on a correctly mapped mirror between the root of your testing site and the root of your deployed site, so if the root doesn’t properly map to subfolders or directories, your links may not work when you go live. After you’ve entered URL information in the field (or browsed to your files), click the Test URL button to see if you have established a connection. If the test is unsuccessful, a number of things could be wrong. Check Appendix A in the section titled “Common Configuration and Deployment Errors” for details.

TIP

When you’re done setting up your site, you may need to go into the Advanced tab in the Site Definition panel and choose Passive FTP in order to create a successful FTP connection, especially if you are working behind a firewall. The unnamed alternative to Passive FTP is called Active FTP and is the default, but it relies on the client’s listening for FTP connections, which doesn’t work well with firewalls.

Using the Advanced Tab to Define a Site

I shy away from wizards whenever I can, because they often make assumptions about my choices that I might later regret. So as quickly as possible, I try to figure out what everything means and understand the options available to me when I’m configuring something on a

Windows machine—whether I’m using Windows itself or an application configuration tool such as Dreamweaver MX’s Site Definition panel.

So let’s hit the Advanced tab in the Site Definition panel and take a closer look at our options (refer again to Figure 1.1 to see where the Advanced tab is). To do this, start the process of going to the Site Definition Wizard outlined at the top of this section immediately under the heading “Using the Site Definition Wizard,” but this time, choose the Advanced tab rather than Basic. Instead of invoking a wizard, you’ll be setting your configuration manually (Site Files > Edit Sites).

So start the process once again by going to Site Files > Edit Sites. In the Edit Sites dialog box, you can choose a current site to edit or you can create a new one. Let’s create a new site and give it whatever name you like. This is where, instead of going into the Basic tab, you should click the Advanced tab. There you’ll see categories on the left, grouped as a column in a tall text box. We’ve covered a lot of these options in the preceding section on using the Site Definition Wizard, so let’s focus here on some details not visible to you when using the wizard.

Local Info

The Local Info category manages the settings pertaining to the machine on which you’re running Dreamweaver. You will normally want to keep images and media in separate directories. This option helps Dreamweaver keep track of them in the File panel’s Assets tab and tends to improve the efficiency of image downloads. (When you use the wizard, you’ll not see a dialog box asking you where to store images.)

Remote Info

This category lets you choose between several options just as the wizard did, except that here you can override what you originally entered into the wizard.

This is where you can choose Passive FTP, as discussed in the earlier Tip. Passive FTP allows the client to initiate FTP connections rather than simply listening for them. For a more technical discussion on this, see the section “The Nefarious ‘Waiting For Server’ Message” in this book’s appendix, “Troubleshooting Web Applications Using Dreamweaver.” You are also presented with an option to automatically upload files when you save the site definition.

Testing Server (and Server Model Language)

Most of the Testing Server options are covered in the section on using the wizard, but here we’ll touch a little more on the significance of choosing your *server model language*.

Since you’re using .NET, this part is pretty easy. I chose to use C# for the examples throughout the majority of this book, but generally the code I use is portable to VB.NET or J#, with

some minor tweaking. (This code is available on the Sybex website, at www.sybex.com, by the way.)

When you choose a server model language, you're basically setting your preferences in such a way that when you click File > New and choose Dynamic Page from the Category window, Dreamweaver MX automatically goes into ASP.NET mode. When you save your first file, Dreamweaver will automatically append the .aspx extension onto the end of your filename, unless you manually override this option.

More importantly, Dreamweaver inserts the correct type of code into your Dreamweaver page when you issue, for example, a Record Insertion Form command from the Dreamweaver menu (Insert > Application Objects > Record Insertion Form), as in this example:

```
<MM:Insert
runat="server"
CommandText='<%# "INSERT INTO chuckwh.AdCopy (AdAutoNumber,
➤ AdTimeStamp, AdDate, AdIO_ID, AdTextCopy, AdTextCopyN, AdImage)
➤ VALUES (@AdAutoNumber, @AdTimeStamp, @AdDate, @AdIO_ID,
➤ @AdTextCopy, @AdTextCopyN, @AdImage)" %>'
ConnectionString='<%# System.Configuration.ConfigurationSettings.AppSettings
➤ ["MM_CONNECTION_STRING_Tumeric"] %>'
DatabaseType='<%# System.Configuration.ConfigurationSettings.AppSettings
➤ ["MM_CONNECTION_DATABASETYPE_Tumeric"] %>'
Expression='<%# Request.Form["MM_insert"] == "form1" %>'
CreateDataSet="false"
Debug="true"
>
  <Parameters>
    <Parameter Name="@AdAutoNumber" Value='<%#
➤ ((Request.Form["AdAutoNumber"] != null) &&
➤ (Request.Form["AdAutoNumber"].Length > 0)) ?
➤ Request.Form["AdAutoNumber"] : "" %>' Type="Int" />
  <!-- SNIP -->
</Parameters>
</MM:Insert>
<MM:PageBind runat="server" PostBackBind="true" />
```

The real work here is performed in a compiled binary object called the `DreamweaverCtrls.dll`, and the `MM:Insert` element is used for interacting with the compiled object and passing parameters to it. This is why Dreamweaver MX can provide a visual interface to robust application server technology. (If you chose VB.NET as your server language, Dreamweaver would output VB.NET code instead.)

NOTE

For more on the `DreamweaverCntnrls.dll`, see the last section of this chapter, “Deploying the Dreamweaver Control,” and Chapter 5, “Working with Components.”

There are a number of additional options in the Testing Server panel designed to help you with site management.

Cloaking

Cloaking lets you hide files from Dreamweaver’s file-uploading services. You provide extensions for the files you don’t want Dreamweaver to upload, such as the native Flash extension, `.fla`, or PSD (Photoshop) files that you may have saved into your Images directory while building site images.

Design Notes

You can pass information to team members using Design Notes, which let you make highly customized notes and add levels of attention for project management purposes. There are two check boxes:

- **Maintain Design Notes.** Enable this to tell Dreamweaver to maintain Design Notes.
- **Upload Design Notes for Sharing.** Enable this if you want to have Design Notes uploaded with the associated file.

Site Map Layout

You can view the structure of your website and the relationship between various site documents using the Site Map Layout tool. In order to use it, you’ll need to choose your home page so that the tool can reveal broken or orphaned links within your site; this is a common feature found in many site management tools.

You can customize the tool by adjusting the number of columns and column width, designating whether icons will contain descriptive text, and choosing whether to display hidden or dependent files.

File View Columns

If you’ve worked with previous versions of Dreamweaver, you might really miss the way the program used to display site files. The old view provided lots of information, such as any associated notes, the date a file was modified and/or checked out, and size. This view is still there, but it’s hidden away in the Site/File panel. When you click the last icon the Site/File panel (Figure 1.3), the panel expands and covers the entire Dreamweaver interface to reveal the “old style” Dreamweaver Site Management window (Figure 1.4).

FIGURE 1.3:
The Site
Management icon

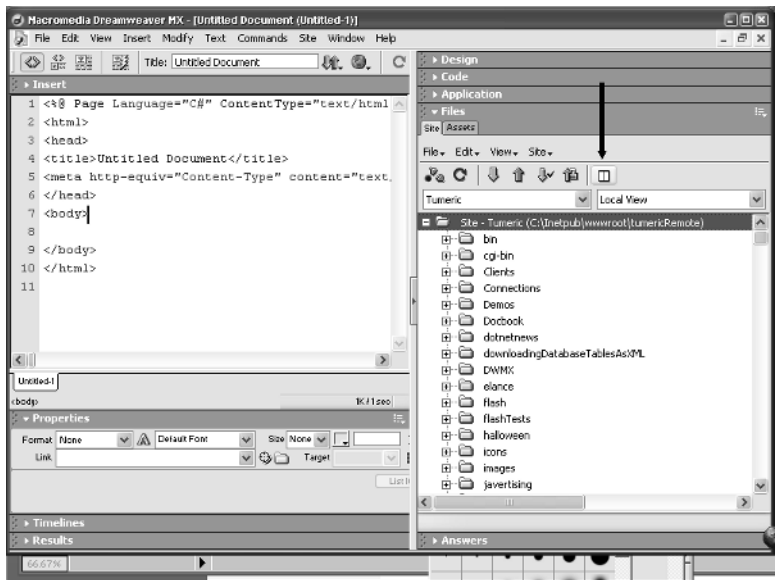
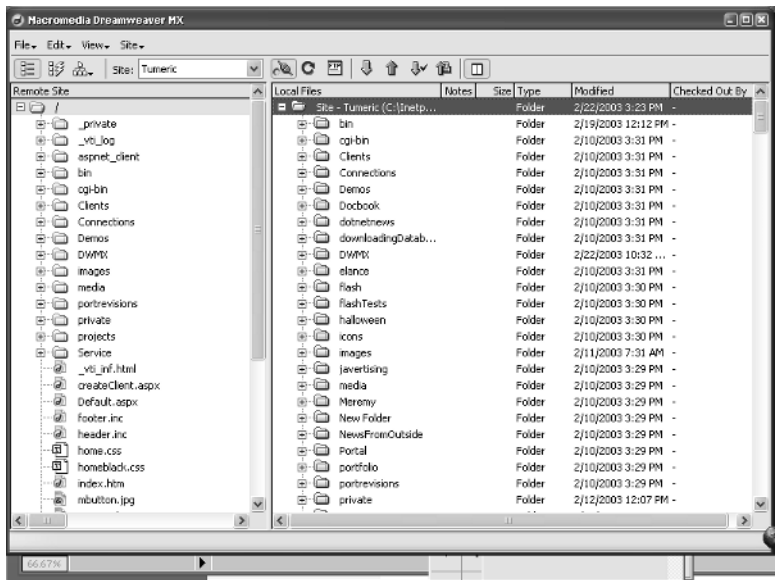


FIGURE 1.4:
The “old-style” Site
Management window
look of the File panel

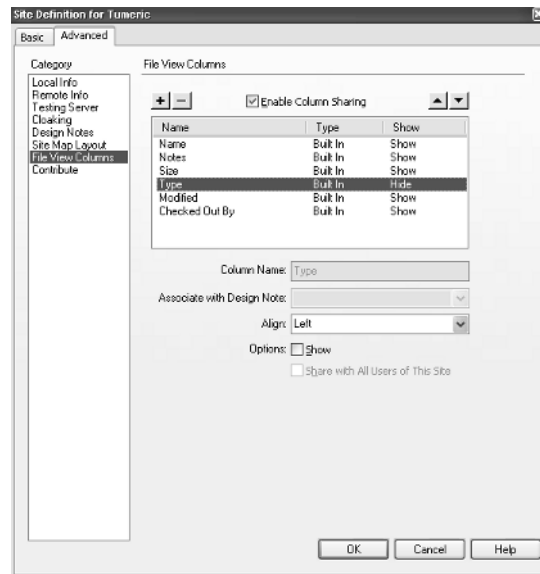


You can customize the look of the old-style view by selecting the next-to-last category in the Advanced tab of the Site Definition panel, File View Columns. By clicking one of the

Options check boxes at the bottom of the File View Columns panel, you can designate whether to hide or show the highlighted feature in the view (Figure 1.5).

FIGURE 1.5:

Highlight a feature and select the Show checkbox to include the item in the old-style File view



Setting Up the IIS Environment for Development

To use ASP.NET with Dreamweaver MX, you will need access to the .NET Framework. Even though .NET Framework is a new technology, it doesn't require an upgrade of Internet Information Services (IIS), which is the Microsoft web server that comes with Windows 2000 and Windows XP. You only need to download the .NET Framework (or get it on CD). You can download it free from Microsoft at <http://www.microsoft.com/net/>.

You also need IIS, which comes with Windows Server operating systems, including Windows 2000 and Windows XP. To test and see if it is already installed, type `http://localhost` into your browser. If you get a "Cannot Find Server" error, you probably need to install IIS. Double-check by opening your Administrative Tools control panel and look for the Internet Information Services component. If you see it, right-click the website you are using by expanding the computer icon in IIS (see Figure 1.6, which shows a highlighted computer within IIS) and then the Web Sites folder. Choose the website you are using and right-click the site. If you're running Windows XP or something other than Windows Professional Server, you will only be licensed for one site, named Default Web Site, so choose that. You'll see a series of options in a context menu. Choose Start. If the website is running, the menu option will say Stop, which means you should actually be seeing the website when you type in `http://localhost` into your browser.

TIP

If you are running XP Home, you'll find that it doesn't install either IIS or Personal Web Server (PWS), the latter of which is a small-footprint web server Microsoft distributed previously with Windows 9x. However, you can go to www.asp.net/webmatrix/default.aspx? and download WebMatrix, which is a free open-source offering by Microsoft. WebMatrix is an editor for .NET files, but it also includes a server that XP Home users can use in lieu of IIS and PWS.

Installing IIS

It's very possible that Internet Information Services (IIS) did not come as part of the installation package when you or someone else installed Windows 2000 or Windows XP Professional on your computer. If that's the case, you'll need to install it.

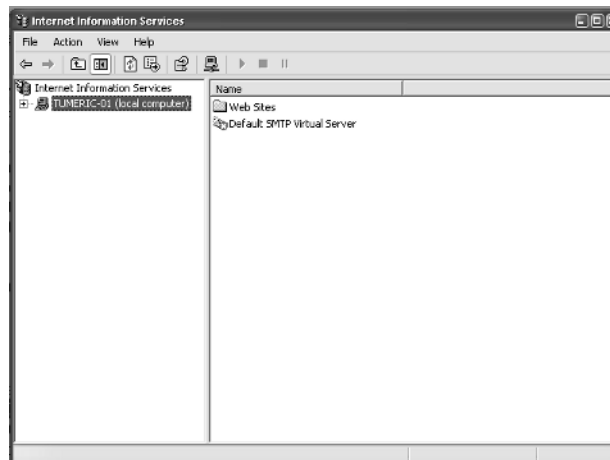
That's easy to do, assuming you have Administrator rights on your computer. If you don't, you'll have to ask your Administrator to set up your system for you.

To install IIS, go to Windows Control Panel and choose Add/Remove Programs. In Windows XP, this option will be named Add/Remove Programs and Windows Components. In Windows 2000, you'll need to drill down one more time to Add/Remove Windows Components. When Windows displays a list of Windows components that can be installed, check the box next to Internet Information Services and click Next to install IIS.

The IIS snap-in will appear in your Administrative Tools Control Panel. If you're running Windows XP and not viewing your Control Panel in classic view, you'll need to click the Performance and Maintenance section of your Control Panel. When you double-click Internet Information Services, you'll see a snap-in expand into a window like the one in Figure 1.6.

FIGURE 1.6:

The IIS snap-in



Configuring IIS

The IIS snap-in works similarly to any other Microsoft Management Console (MMC) window. MMC is a hosting environment that presents a common interface for multiple applications such as IIS and Index Server. This makes it easier for users to use what really amounts to a number of fairly complex Windows components, without needing to figure out the nuances of the assortment of interfaces for those components. Even if you've never used an MMC-based component, it's easy to learn because it looks so much like Windows Explorer.

At the top of the IIS MMC window is an icon and the name Internet Information Services. Under that is your computer name, and next to that is a little plus sign, just like Windows Explorer. When you click the plus sign, the computer expands to reveal the components of your computer that are relevant to IIS. One of those will be a folder named Web Sites. Expand Web Sites and you'll see your Default Web Site. If you have Windows 2000 Server, you can add additional sites. If you have what amounts to personal editions of Windows 2000, such as Windows 2000 Professional or Windows XP, you won't be able to add additional web-sites and you'll have to use the Default Web Site because that's all you're licensed for.

You're actually ready to go, but chances are you'll want to break your web applications down into separate directories. This means you need to make a *virtual directory*. A virtual directory is a mapped directory interface to a physical directory on the web server's computer. By creating a virtual directory, you can have a directory located at

```
C:/documents/docs/moredocs/etmoredocs/docs/xdocs/ydocs/vdocs/
```

accessed via the server at

```
localhost/vdocs
```

This is because you tell the web server that the physical path should point to the virtual directory mapping you provide.

To try it out, right-click the Default Web Site in IIS and navigate to New/Virtual Directory. The New Virtual Directory Wizard will appear. The first thing you'll have to do is name your virtual directory by entering it into the Alias text field. The name you give is the name of your web application. You can have more than one web application in a web-site. In fact, you can have zero web applications in your website and run everything from the root (and still have subdirectories), but that's not a terribly efficient way to do things.

The next page asks you to locate the physical folder that stores the files to be used by the virtual directory. It's important to realize at this point that the Virtual Directory doesn't actually store any files, or even copy them anywhere, even though it will look like it does when you're finished with the wizard. The Virtual Directory is an application that runs by implementing code it finds in the physical directory you name on this page of the wizard. The nice thing about this is that, even if you serve an HTML page from a directory that is

not involved in any way with your virtual directory, you can refer to the virtual directory in any code on your site like this (assuming our current example):

```
<a href = "/vdocs/foo.html">A link</a>
```

Notice where the first solidus (/) character is on the path. Using that character as the first character on a path tells IIS to look for a virtual directory named by whatever lies between the first and second solidus characters. So in the preceding line of code, the link points to a virtual directory named vdocs that would resolve to the domain name followed by vdocs, or

```
http://www.domainname.com/vdocs
```

Another advantage to using virtual directories is that you can make them resolve to subdomains like this:

```
http://services.domainname.com/vdocs
```

where vdocs resolves to the subdomain services.

The last page of the wizard asks you which Access Permissions you want for your Virtual Directory. For our purposes here, you will generally not need anything more than the Read and Run Scripts permissions. Exceptions to this will be noted during the course of the book. Always make Read and Run Scripts your choice as your instinctive reaction (rule of thumb) to this page. You can always change things later.

NOTE

The configuration described in this section was tested on Windows XP Professional. Your IIS configuration windows may differ slightly but will be basically the same.

Configuring Dreamweaver MX for Use in Versioning Software

Versioning software, also referred to as *source control* or *content management systems software*, is software that locks files when a team member checks them out for editing so that team members can't edit the same files simultaneously. Among such software, Dreamweaver MX provides direct support only for Microsoft's Visual SourceSafe and Macromedia's SiteSpring. SiteSpring is being left to die on the vine by Macromedia; they have released their last version of the product.

Before I introduce you to specific source control systems, let's have a brief look at team workflow as practiced within the Dreamweaver environment.

Checking In and Checking Out

The basic tenet of Dreamweaver's Check In/Check Out system is based on one thing: Don't use the Get or Put commands, ever, if you are using Check In/Check Out. Get and Put are

FTP commands that get files from the server and put files onto the server, respectively. Of course, there might be exceptions to this if you are using, for example, Perforce, which is a third-party source control system, but that's because Perforce has its own checking-in-and-out workflow that bypasses Dreamweaver's. (There's more about configuring Dreamweaver MX for Perforce in a later section.)

NOTE

To use Check In/Check Out, each team member must be using Dreamweaver, which can be a drawback if you have programmers who resist using it. This is one of the reasons it's sometimes best to develop using a third-party source control system. If every team member is using Dreamweaver MX, Check In/Check Out can be a good option for version control, but keep in mind that it doesn't lock files as true source control software does.

Team Workflow

To use Check In/Check Out, each team member defines a local root folder on his or her development machine. If you're an administrator, configure a common remote sharing server. Otherwise, your administrator will do that for you. Make sure that in your Site Definition panel, under Remote Server in the Advanced tab, you check the Enable File Check In and Check Out box. Then, to retrieve a file, instead of using the Get command, you use the Check Out command. All team members access files this way; so when a file is checked out, it's critical that you do not check it out. You can tell by viewing the icons in the File panel that indicate whether or not files are checked out (refer again to Figure 1.4).

The bad news about Check In/Check Out is that the system doesn't lock the files in the same way as industrial-strength source control systems, so it's actually possible to check out a file even if it has already been checked out, and then overwrite someone else's changes when you check the file back in. This can also happen if you use the Get and Put commands—a good reason never to do this when you're using Check In/Check Out. Users attempting to retrieve files that were checked out by someone else will get a confirmation message asking if they really want to go through with it. Even knowledgeable users, however, can make mistakes and accidentally hit the OK button, especially when they're distracted.

If it sounds like I'm arguing that workgroups should design a better system, one that really locks files when they're checked out, I am. Nevertheless, Check/In Check Out is available in Dreamweaver, and there may be some instances where you might use it. So let's take a quick look at how to set it up.

Setting Up Check In/Check Out

You must first define a site and select certain options in the Site Definition panel to activate Check In/Check Out:

Setting up the local site Each team member defines a site, designating a folder on their computer's hard drive as the local root folder. You already know how to do this from the section titled "Defining a Site" earlier in the chapter.

Setting up the remote site In the Site Definition dialog panel, in the Advanced Tab's Remote Info category, specify a server folder (Host Directory for FTP access) or a remote folder (for local/network access) as shown in the section titled "Defining a Site" earlier in the chapter.

Enabling Check In/Check Out In the Remote Info category, select the Enable Check In/Check Out check box. Two more text fields appear beneath that check box. Fill in a name and your e-mail address in the two fields provided. After these options have been activated, the Check In and Check Out icons appear in the File panel.

Configuring for Concurrent Versions System (CVS)

Concurrent Versions System (CVS) is a reliable old workhorse among long-time developers. On Windows the most commonly used implementation is WinCVS. On the Mac, you can get a CVS implementation called MacCVS. WinCVS has proven frustrating for most Dreamweaver developers in production environments, but there are two options to consider to make it a viable solution for versioning.

The first is a SourceForge Dreamweaver/CVS project that is in its earliest stages for integrating Dreamweaver with WinCVS. You'll find it at <http://dwcvs.sourceforge.net/>.

NOTE

As this book was being written, the SourceForge project was in its earliest stages, and the developers had not yet posted any files. That may have changed by the time you read this.

A better bet, because it gets good reviews for stability and ease of use and is in fact available, is another SourceForge project named TortoiseCVS. There isn't any specific module for integration with Dreamweaver, but as a general versioning control system, it's worth a try: <http://www.tortoisecvs.org/download.shtml>. And MacCVS is available at SourceForge at <http://www.maccvs.org/>.

You'll have to use TortoiseCVS and MacCVS outside of Dreamweaver because direct integration with them isn't supported.

Configuring for Perforce

Perforce is a versioning system that has a loyal following in many large-scale development shops. You can configure Dreamweaver to work with Perforce's storage units (called depots) and access them in Dreamweaver using FTP. The following instructions assume a basic knowledge of Perforce.

There are three Perforce directory areas to be aware of:

- The client workspace is an area of working storage on the computer where P4FTP, the Perforce FTP tool, runs. This workspace must not overlap the local root.
- The local root is the directory where Dreamweaver stores its working copies of the website files for editing.
- The host directory is the Perforce depot location where your website files live. The remote host directory must be the path to your Perforce ftp- client workspace, plus any additional directories that you need in order to get you to the files you're working on.

For example, suppose you're running Dreamweaver on a Windows computer, P4FTP on a Unix computer named ftpserver, and the Perforce server on a Unix computer. You would use the following Perforce configuration:

Host: ftpserver

Client root: /usr/team/userName

Client view: //depot/...//ftp-userName/...

The Dreamweaver configuration for a website named mySite would be as follows:

Local root: c:\dwroot\mySite

Host directory: /usr/team/userName/main/mySite/

Creating Websites

To use Dreamweaver to create a website that resides in a Perforce depot, follow these steps:

1. Choose **➤ New Site**. The Site Definition dialog box is displayed. Click the Advanced Tab.
2. Create a folder on your computer where you want Dreamweaver to store the local copies of your website files, making sure you specify an existing directory that does not live in your client root.
3. In the Category pane, click Remote Info. The Remote Info pane is displayed.
4. From the Access list, choosing FTP reveals the following fields:

FTP Host Enter the name of the computer where P4FTP is running and the port on which it is listening for FTP requests. For example: **myftphost:1234**.

Host Directory This is the location in the Perforce depot to upload and download website files using the Put and Get FTP commands. If you're running in a mixed OS environment, the name of this directory should be based on the conventions of the operating

system that is running P4FTP. This must be an existing directory, so if one doesn't exist, create one first using Perforce or through an FTP client with a command-line interface.

Login and Password Enter your Perforce login and password.

WARNING Do not check the box Enable File Check In and Check Out.

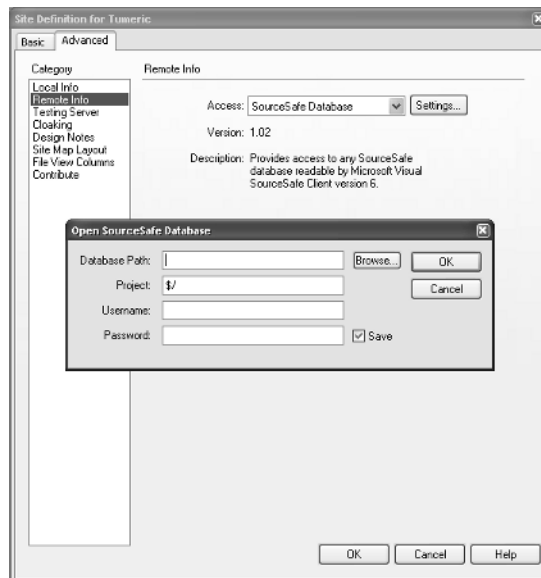
The next step is to create the Perforce depot location in your client workspace. Copy any existing website files to the client workspace location that corresponds to the depot location of the website you defined with the Site Definition panel. Use the Perforce p4 Add command to add files to the depot.

Configuring for Visual SourceSafe

Dreamweaver MX provides direct support for Visual SourceSafe. You can connect directly to SourceSafe through the Remote Info category in the Advanced tab of the Site Definition panel. When you choose SourceSafe Database from the Access drop-down list, a Settings button appears to the right of the drop-down list. Click that button to display a dialog box with the settings you need to configure for connecting to the desired Visual SourceSafe database (see Figure 1.7). After you complete these entries, you can connect to the SourceSafe database and check files in and out.

FIGURE 1.7:

The Settings input for connecting to Visual SourceSafe



It's possible that the site administrator won't want direct connections to the Visual SourceSafe database, in which case you'll have to work on your site locally and check files in and out of Visual SourceSafe using your Working folder in SourceSafe. Details on this will depend on how your administrator has set up the SourceSafe environment, so be sure to check with your site administrator for details.

NOTE

Unfortunately, up to this point, my experience with WebDAV in Dreamweaver has not been very positive. In fact, every time I have attempted to use it, the application has crashed, and this has been the case on multiple versions of Dreamweaver MX and Dreamweaver MX 2004.

The Art of Managing the *Web.config* File

Anytime you're working with component-based architecture, and .NET as deployed from Dreamweaver certainly qualifies, it's best to think about how many problems can be solved globally. By this I don't mean solving the world's problems, but rather how many of your web development tasks can be resolved on a global level, so that you can drill down and add specificity on a granular level.

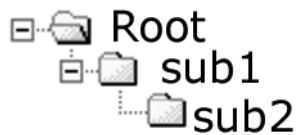
To see what I mean, pay special attention to this section, because the more you can manage configuration outside of your actual application code, the better. In ASP.NET, you can control a number of your web application's configuration properties through an XML file named `Web.config`.

When you store a `Web.config` file in your domain's root directory, all the files throughout the rest of your site inherit the properties that are in the configuration file. If there is no `Web.config` file, the application acts as if you have one in the root anyway, and everything is defaulted to ASP.NET's settings that the Framework adopted when it was originally installed.

The settings of the `Web.config` file, whether they're placed by you or are the .NET default settings, can be overridden by adding a configuration file named `Web.config` to the directory that contains the application you want to influence. Looking at Figure 1.8, if you want to make some application configurations in the directory named `Sub1` different from the rest of your site, you would add a `Web.config` file to `Sub1`. The web applications in `Sub2`, since it is a child of `Sub1`, would then inherit the configurations deployed in `Sub1`. So if you wanted them to be like the rest of the site, you'd need to either provide that directory with its own `Web.config` file, or set up different `system.web` elements for each directory within the root directory's `Web.config` file.

FIGURE 1.8:

Directories inherit
Web.config
properties from
their parents



Listing 1.1 shows what a typical structure for a Web.config file might look like.

Listing 1.1 **Typical structure for a Web.config file**

```
<configuration>
  <appSettings />
  <system.web>
    <authentication>
      <forms>
        <credentials>
        <passport>
      </authentication>
    <authorization>
      <allow>
      <deny>
    </authorization>
    <browserCaps>
      <result>
      <use>
      <filter>
        <case>
      </filter>
    </browserCaps>
    <clientTarget>
      <add>
      <remove>
      <clear>
    </clientTarget>
    <compilation>
      <compilers>
        <compiler>
      </compilers>
      <assemblies>
        <add>
        <remove>
        <clear>
      </assemblies>
    </compilation>
    <customErrors>
      <error>
    </customErrors>
    <globalization>
    </globalization>
    <httpHandlers>
      <add>
      <remove>
      <clear>
    </httpHandlers>
    <httpModules>
      <add>
      <remove>
    </httpModules>
  </system.web>
</configuration>
```

```
<clear>
<httpRuntime>
<identity>
<machineKey>
<pages>
<processModel>
<securityPolicy>
  <trustLevel>
</securityPolicy>
<sessionState>
<trace>
<trust>
<webServices>
  <protocols>
    <add>
    <remove>
    <clear>
  </protocols>
  <serviceDescriptionFormatExtensionTypes>
    <add>
    <remove>
    <clear>
  </serviceDescriptionFormatExtensionTypes>
  <soapExtensionTypes>
    <add>
  </soapExtensionTypes>
  <soapExtensionReflectorTypes>
    <add>
  </soapExtensionReflectorTypes>
  <soapExtensionImporterTypes>
    <add>
  </soapExtensionImporterTypes>
  <WSDLHelpGenerator>
</webServices>
</system.web>
</configuration>
```

You may want to configure everything in one root directory, however. It certainly will be easier to maintain, because you won't have to remember to update various `Web.config` files as your site grows. Instead, you just need to remember to alter the root `Web.config` file.

For example, you might want to create particular configurations for different languages. In that case, you wrap each `system.web` element in a `location` element, like so:

```
<location path="EnglishPages">
  <system.web>
    <globalization
      requestEncoding="iso-8859-1"
      responseEncoding="iso-8859-1"
    />
  </system.web>
</location>
```

```

<location path="JapanesePages">
  <system.web>
    <globalization
      requestEncoding="Shift-JIS"
      responseEncoding="Shift-JIS"
    />
  </system.web>
</location>

```

Here, a different `system.web` configuration element is used for Japanese and English pages. All files in the `EnglishPages` directory will use the `iso-8859-1` encoding, and all files living in the `JapanesePages` directory will use `Shift-JIS` encoding. You can also use any of the other `system.web` child elements that you'll be visiting in the next section.

Web.config Elements and Attributes

It doesn't do much good to have this nifty XML-based configuration document if you don't know what its elements and attributes mean. This section provides a look into the properties of the one file you're likely to encounter the most while working with Dreamweaver. We're not providing a detailed analysis of every element and attribute available to this file, because, like so many Microsoft object models, the underlying schema that defines these elements is substantial and there just isn't enough space in this book to justify the deforestation such detailed analysis would trigger. Instead, I've focused on areas that will be of particular interest to Dreamweaver MX .NET developers. You can get more information from the .NET Framework SDK at

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/gnrgfaspnnetconfigurationsectionschema.asp>

appSettings

One of the main reasons for using Dreamweaver MX in the first place is to take advantage of its rapid application development (RAD) capabilities, especially when you're working with databases. If you look at your site's `Web.config` file after you set up your database, you'll find something like this:

```

<appSettings>
  <add key="MM_CONNECTION_HANDLER_MyConnection" value="sqlserver.htm" />
  <add key="MM_CONNECTION_STRING_MyConnection" value=
    "Persist Security Info=False;Data Source=11.11.1.1;
    Initial Catalog=mydb;User ID=user;Password=somepass" />

```

```

    <add key="MM_CONNECTION_DATABASETYPE_ MyConnection" value="SQLServer" />
    <add key="MM_CONNECTION_SCHEMA_ MyConnection" value="" />
    <add key="MM_CONNECTION_CATALOG_Tumeric" value="" />
  </appSettings>

```

These settings are created by Dreamweaver and inserted into your root directory's Web.config file when you set up your first database connection. The settings, of course, will depend on exactly what settings you choose when you create your database connection string to connect to your database. The process is driven by the appSettings element, which can have three child elements:

The add element, which is the one you'll see used most often, has two attributes, key and value. The key is simply the name of the application setting that will be accessed by .NET during runtime, and the value is the value sent to .NET to process the component. For example, when you create a dataset in Dreamweaver MX, Dreamweaver inserts a custom .NET control element named MM:DataSet into your web page. One of the attributes for this element is the ConnectionString attribute. In Figure 1.9 you can see there is a direct connection between the MM:DataSet element's ConnectionString attribute and the Web.config file's element's appSettings element. The ConnectionString calls the key named MM_CONNECTION_STRING_Tumeric, and that key is the value of the connection string. Therefore, this:

```

ConnectionString='<%# System.Configuration.ConfigurationSettings.AppSettings
➡ ["MM_CONNECTION_STRING_MyConnection"] %>'

```

could have been written like this directly in an .aspx page, instead:

```

ConnectionString="Persist Security Info=False;Data Source=11.11.1.1;
➡ Initial Catalog=mydb;User ID=user;Password=somepass"

```

This form, however, forces us to edit the connection string in the .aspx page that is using it. And we don't want to do this, because if we end up with numerous such pages we'd have to go into each page in order to edit the connection string. It's much easier to define the connection string within the appSettings element and refer to it repeatedly in other pages. That way, if we want to change it later, we easily can by simply updating the Web.config file.

The remove element of appSettings can be used to remove a configuration setting deployed further up in the configuration hierarchy.

The clear element removes all settings defined further up in the hierarchy.

FIGURE 1.9:

The `ConnectionString` attribute in `MM:DataSet` is bound to the second `add` element child of the `appSettings` element.

MM:DataSet element

```
<MM:DataSet
  id="clientDS"
  runat="Server"
  IsStoredProcedure="false"
  ConnectionString='<%#
System.Configuration.ConfigurationSettings.AppSettings["MM_CONNECT
ION_STRING_myConnection"] %>'
  DatabaseType='<%#
System.Configuration.ConfigurationSettings.AppSettings["MM_CONNECT
ION_DATABASETYPE_myConnection"] %>'
  CommandText='<%# "SELECT Client_ID FROM ClientInfo WHERE CEMail =
@CEMail" %>'
  Debug="true"
>
```

appSetting element in Web.config file

```
<appSettings>
  <add key="MM_CONNECTION_HANDLER_myConnection"
value="sqlserver.htm" />
  <add key="MM_CONNECTION_STRING_myConnection" value="Persist
Security Info=False;Data Source=11.111.1.11;Initial Catalog=myDB;User
ID=uid;Password=yourpassword" />
  <add key="MM_CONNECTION_DATABASETYPE_myConnection"
value="SQLServer" />
  <add key="MM_CONNECTION_SCHEMA_myConnection" value="" />
  <add key="MM_CONNECTION_CATALOG_myConnection" value="" />
</appSettings>
```

authentication

The `authentication` element configures ASP.NET authentication support. This is one of the more important elements when you are building Dreamweaver .NET apps, because it makes it easy to develop password-protected areas and redirection routines. Authentication can be handled in one of four ways, as represented by the `mode` attribute:

- Through Windows authentication
- Through forms
- Through Microsoft's proprietary Passport system
- None (there is no authentication)

The syntax for this element looks like this:

```
<authentication mode="Windows|Forms|Passport|None">
  <forms name="name"
    loginUrl="url"
    protection="All|None|Encryption|Validation"
    timeout="30" path="/" >
    <credentials passwordFormat="Clear|SHA1|MD5">
      <user name="username" password="password" />
    </credentials>
  </forms>
  <passport redirectUrl="internal" />
</authentication>
```

The authentication element has some child elements that are worth looking at. These all help you develop robust authentication routines for directories, virtual directories and applications.

The *forms* Element

The forms child element has nothing to do with the HTML form element we've all grown to love and nurture. Rather, this is used for the .NET authentication process. Consider this element as the key to the city for your users. Let's say you set up a virtual directory or web application named `catalog` that is accessed when a user types `www.mydomain.com/catalog` into their browser. You use the forms element to handle the users' routing so that everyone, if you want, goes to the same page. This means if someone has a part of this directory bookmarked, let's say `www.mydomain.com/catalog/profile.aspx`, they can only get there if they are an authenticated user. You use the forms element to manage this routing apparatus.

There are a number of attributes to the forms element you might want to get familiar with, so let's have a look:

name This attribute specifies the HTTP cookie to use for authentication. By default, the value of name is `.ASPXAUTH`. When more than one application is running on a single server and each application requires a unique cookie, each application must have its own cookie name in each application's Web.config file.

loginUrl When a user attempts to visit a page on a site that has been configured for authentication, the user will first get redirected to the page specified by the loginUrl attribute if no valid authentication cookie is found; this happens even if the page is bookmarked. The default value is `default.aspx`, but you can change it to any name you wish. The page to which you redirect the users should have a login form, or, at a bare minimum, a link to a login form, since the page really acts as the gateway to the rest of the directory configured for authentication.

protection This attribute specifies what kind of encryption to use. There are four possible values for this attribute:

- **All:** The default and recommended value; this means the application uses both data validation and encryption to protect the cookie.
- **None:** Although it makes authentication less challenging to system resources, Microsoft recommends against using the none protection value because both encryption and validation are disabled.
- **Encryption:** Specifies that the cookie is encrypted using Triple-DES or DES (for more information on encryption schemes, go to <http://csrc.nist.gov/cryptval/des.htm>), but data validation is not performed on the cookie. According to Microsoft, cookies used in this way might be subject to chosen plaintext attacks.
- **Validation:** This is a validation confirmation technique for preventing cookie data tampering while the data is transported over the network. This attribute value tells .NET to create a cookie by concatenating (adding one string value to another to form a new string) a validation key with cookie data, computing a message authentication code, and appending the message authentication code to the outgoing cookie.

timeout This specifies the amount of time, in integer minutes, after which the cookie expires. The default value is 30.

path Use the path attribute to specify the path for cookies generated by your application. It's best to keep the path at the default value, which is a backslash (\), because most browsers are case-sensitive and will not send cookies back if there is a case mismatch in the path.

The forms child element itself has an optional child element named `credentials`, consisting of a mandatory attribute named `passwordFormat` that you use to indicate how you want to encrypt the password. You have three options: `Clear`, which provides no encryption, and which you obviously want to avoid unless you want hackers to have access to your users' passwords; `MD5`, for the MD5 hash algorithm; and `SHA1`, for the SHA1 hash algorithm.

The `credentials` element, too, has a child element named `user`, which can be used to store a username and password directly in the `Web.config` file.

Take a look at the following `Web.config` file snippet, a form that authenticates against an application named `logon.aspx`. This form is the gateway, so to speak, for all who attempt to access a file in the directory. If users try to access a file but hasn't yet logged on, they are automatically re-directed to `logon.aspx`.

```
<authentication mode="Forms">
  <forms name=".ASPXAUTH"
    loginUrl="logon.aspx"
    protection="All" path="/" timeout="30" />
</authentication>
```

Listing 1.2 is an example of the script that drives logon.aspx. When a user tries to get into another page in the same directory, if they haven't logged in, they'll get bounced back to logon.aspx. This is true even if they know the query string that would take them to their profile page and try to type that into their browser.



Listing 1.2 A login script for logon.aspx

```
<script runat="server">
    bool ValidateUser(string uid, string passwd)
    {
        SqlConnection cnn;
        SqlCommand cmd;
        SqlDataReader dr;
        cnn = new SqlConnection(ConfigurationSettings.AppSettings
➤ ["MM_CONNECTION_STRING_Tumeric"]);
        cmd = new SqlCommand("SELECT Client_ID, CPassword, ClientName,
➤ CFirstName,
➤ CLastName FROM ClientInfo where CEMail='" + uid + "'",cnn);
        cnn.Open();
        dr = cmd.ExecuteReader();
        try {
            while (dr.Read())
            {
                if (string.Compare(dr["CPassword"].ToString(),passwd,false)==0)
                {
                    cnn.Close();
                    return true;
                }
            }
        }
        catch (Exception e)
        {
            lblMsg.Text = "An application error has occurred. Please call us
➤ at 415-553-8857 for further assistance.";
        }
        cnn.Close();
        return false;
    }

    void Page_Load(object sender, System.EventArgs e)
    {
        if (Page.IsPostBack)
        {
            if (ValidateUser(txtUserName.Value,txtUserPass.Value) )
            {
                FormsAuthenticationTicket tkt;
                string cookiestr;
                HttpCookie ck;
                tkt = new FormsAuthenticationTicket(1, txtUserName.Value,
➤ DateTime.Now,
➤ DateTime.Now.AddMinutes(30), chkPersistCookie.Checked,
```

```

        ➤ "your custom data");
        cookiestr = FormsAuthentication.Encrypt(tkt);
        ck = new HttpCookie(FormsAuthentication.FormsCookieName, cookiestr);
        if (chkPersistCookie.Checked)
            ck.Expires=tkt.Expiration;
        Response.Cookies.Add(ck);

        string strRedirect;
        strRedirect = Request["ReturnUrl"];
        if (strRedirect==null)
            strRedirect = "client2.aspx?CEMail=" + txtUserName.Value;
        Response.Redirect(strRedirect, true);
    }
    else
        lblMsg.Text = "You have provided an invalid user name or password.
        ➤ Please try again";
    }
}

```

You'll learn how to build this kind of form authentication in Chapter 4, "Working with Databases."

authorization

You can also allow users entry into a directory or application through the authorization element and its child allow and deny elements, which define who gets in and who doesn't.

```

<authorization>
    <allow users="comma-separated list of users"
        roles="comma-separated list of roles"
        verbs="comma-separated list of verbs" />

    <deny users="comma-separated list of users"
        roles="comma-separated list of roles"
        verbs="comma-separated list of verbs" />
</authorization>

```

When defining who gets in and who doesn't, a question mark (?) allows anonymous users; an asterisk (*) indicates that all users are accepted, as in this next example. It allows access to all members of the Admin role and denies access to everyone else by blocking out all users using an asterisk.

```

<configuration>
    <system.web>
        <authorization>
            <allow roles="Admins" />
            <deny users="*" />
        </authorization>
    </system.web>
</configuration>

```

In authorization code, users and roles may be obvious to you, but what about verbs? This attribute is a comma-separated list of HTTP methods granted access to the resource. Possible values are GET, HEAD, POST, and DEBUG.

browserCaps

The browserCaps exposes ASP.NET's browser capabilities component to help you better handle various incoming browsers. This lets you write code to make your web application behave differently depending on the browser type accessing your site. This process is called *browser sniffing*, often done in the past using client-side JavaScript. The browserCaps element structure looks like this:

```
<browserCaps>
  <result type="class" />
  <use var="HTTP_USER_AGENT" />
    browser=Unknown
    version=0.0
    majorver=0
    minorver=0
    frames=false
    tables=false
  <filter>
    <case match="Windows 98|Win98">
      platform=Win98
    </case>
    <case match="Windows NT|WinNT">
      platform=WinNT
    </case>
  </filter>
  <filter match="Unknown" with="%(browser)">
    <filter match="Win95" with="%(platform)">
    </filter>
  </filter>
</browserCaps>
```

The filter element's with attribute uses regular expressions, the nuances of which are beyond the scope of this chapter, but generally you won't need to alter the default capabilities of ASP.NET's browser-sniffing. You can use .NET's `HttpBrowserCapabilities` class to manage the web application behavior based on the kind of browser attempting to access the page.

Back in the days of the browser wars, redirecting users using different browsers was one way, if an expensive one, to manage browser capabilities. Generally, today's developers shy away from expensive browser-sniffing routines because most modern browsers interpret standard HTML in a reliable way, and you can use Cascading Style Sheets (CSS) to enhance visual layouts. Browsers that don't support CSS won't display CSS-based visual enhancements, but if the coding is done correctly, these browsers will still display the page reasonably well because they understand the basic HTML elements that to render the pages.

Nevertheless, you may find yourself wanting or even needing to do some browser sniffing, especially if you're on a corporate intranet and you want to take advantage of, for example, IE6's extensive Dynamic HTML model. To make procedural decisions based on browser make and version, you can do something like this:

```
if(Request.Browser.Browser.Equals("IE") && Request.Browser.MajorVersion >= 5)
    Response.Write("Good, you are using IE 5 or higher");
```

clientTarget

.NET contains a collection of user agent aliases, which bind arbitrary names (for example, "ie5") to user agent information. You can add to this collection using your own aliases. Use whatever name you want in the alias attribute, but the userAgent attribute to which you bind the name needs to be recognizable by .NET, such as

```
Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)
```

Here's an example of clientTarget code:

```
<clientTarget>
  <add alias="alias name to use"
        userAgent="identification of user agent" />
  <remove alias="alias name to remove" />
  <clear />
</clientTarget>
```

compilation

This is an element that contains all the compilation settings used by ASP.NET:

```
<compilation debug="true|false"
              batch="true|false"
              batchTimeout="number of seconds"
              defaultLanguage="language"
              explicit="true|false"
              maxBatchSize="maximim number of pages per
                           batched compilation"
              maxBatchGeneratedFileSize="maximum combined size
(in KB) of the generated source file per batched compilation"
              numRecompilesBeforeAppRestart="number"
              strict="true|false"
              tempDirectory="directory under which the ASP.NET temporary
                           files are created" >
  <compilers>
    <compiler language="language"
              extension="ext"
              type=".NET Type"
              warningLevel="number"
              compilerOptions="options" />
  </compilers>
```

customErrors

This element defines custom error messages for an ASP.NET application:

```
<customErrors defaultRedirect="url"
              mode="On|Off|RemoteOnly">
  <error statusCode="statusCode"
        redirect="url" />
</customErrors>
```

If you wanted to redirect a user who was receiving a 404 File Not Found error, you could do something like this:

```
<customErrors defaultRedirect="generr.htm"
              mode="RemoteOnly">
  <error statusCode="404"
        redirect="FNF.htm" />
</customErrors>
```

You should turn custom errors on by making the mode value `On` when you are in development, and turn them off when you are ready to go live by making the mode value `RemoteOnly`. Actually, `RemoteOnly` doesn't turn them off; it redirects the user to a custom error page so that they don't see the ASP.NET errors that are of benefit to you as a Web developer.

globalization

You encountered this element when introduced to the `Web.config` element earlier in the chapter. The globalization element configures the globalization settings of an application:

```
<globalization requestEncoding="any valid encoding string"
               responseEncoding="any valid encoding string"
               fileEncoding="any valid encoding string"
               culture="any valid culture string"
               uiCulture="any valid culture string" />
```

You might be wondering what the culture-related strings do. They provide ways to automatically handle things like regional-specific date data-types. Dates in the United States are typically structured as "August 11, 2003," whereas dates in some other countries are typically structured as "11 August 2003." In addition, .NET sets the default currency to euro (instead of the dollar) for cultures that use the euro.

httpHandlers

This element maps incoming URL requests to `IHttpHandler` classes:

```
<httpHandlers>
  <add verb="verb list"
        path="path/wildcard"
        type="type,assemblyname"
        validate=" " />
```

```
<remove verb="verb list"
      path="path/wildcard" />
<clear />
</httpHandlers>
```

Handlers accept requests and produce responses and manage them through page-handler classes. One area where you might see this `httpHandlers` element used frequently is Web Services, which allows a site to expose programmatic functionality via the Internet by accepting messages and sending replies, and running complex functions based on this communication.

When the .NET runtime environment sees a request for a file with an `.aspx` extension, the handler that is registered to handle `.aspx` files is called, which by default is the `System.Web.UI.PageHandlerFactory` class. However, when an `.asmx` file is detected, Web Services are invoked through a different class, the `System.Web.Services.Protocols.WebServiceHandlerFactory`. By customizing this process through the `Web.config` file, you can determine which HTTP requests are handled by using the `verb` attribute of the `add` child element of `httphandlers`.

```
<httphandlers>
<add verb="GET" path="*.asmx"
type="System.Web.Services.Protocols.WebServiceHandlerFactory,
System.Web.Services" validate="false" />
</httphandlers>
```

This code designates that for all GET requests, if the file being requested is an `.asmx`, the system should create an instance of the `WebServiceHandlerFactory`, which lives in the `System.Web.Services.dll` assembly. If you want the handler to accept all HTTP verbs, you would change the `verb` value to `"*"`. This makes it possible to generate an HTML page that describes a Web Service.

httpModules

The `httpModules` element adds, removes, or clears HTTP modules within an application. Most of the processes you encounter using Dreamweaver .NET have everything you need in the default settings. The `httpModules` element has three subelements: `add`, `remove`, and `clear`. The `add` subelement has one attribute, `type`, which is the assembly and class name of the handler you wish to include:

```
<httpModules>
  <add type="classname,assemblyname" name="modulename" />
  <remove name="modulename" />
  <clear />
</httpModules>
```


httpRuntime

This element configures ASP.NET HTTP runtime settings. This section can be declared at the machine, site, application, or subdirectory level:

```
<httpRuntime useFullyQualifiedRedirectUrl="true|false"
    maxRequestLength="size in kbytes"
    executionTimeout="seconds"
    minFreeThreads="number of threads"
    minFreeLocalRequestFreeThreads="number of threads"
    appRequestQueueLimit="number of requests" />
```

identity

Normally, all ASP.NET application operations are performed by a user named ASPNET. This user is created automatically when you install the .NET Framework. Then, you have to be sure the virtual directory (your web application) has been granted user privileges to ASP.NET. Sometimes you'll want to "impersonate" a user, which means that instead of using the ASPNET user account, you'll name another account for running the web application. The identity element controls this process with some fairly self-explanatory attributes:

```
<identity impersonate="true|false"
    userName="username"
    password="password" />
```

machineKey

This element configures keys to use for encryption and decryption of a form's authentication cookie data and can be declared at the machine, site, or application levels, but not at the sub-directory level:

```
<machineKey validationKey="autogenerate|value"
    decryptionKey="autogenerate|value"
    validation="SHA1|MD5|3DES" />
```

pages

The page element identifies page-specific configuration settings such as session state and page buffering:

```
<pages buffer="true|false"
    enableSessionState="true|false|ReadOnly"
    enableViewState="true|false"
    enableViewStateMac="true|false"
    autoEventWireup="true|false"
    smartNavigation="true|false"
    pageBaseType="typename, assembly"
    userControlBaseType="typename" />
```

processModel

The `processModel` element configures the ASP.NET process model settings on Internet Information Services (IIS) web server systems:

```
<processModel enable="true|false"
  timeout="mins"
  idleTimeout="mins"
  shutdownTimeout="hrs:mins:secs"
  requestLimit="num"
  requestQueueLimit="Infinite|num"
  restartQueueLimit="Infinite|num"
  memoryLimit="percent"
  cpuMask="num"
  webGarden="true|false"
  userName="username"
  password="password"
  logLevel="All|None|Errors"
  clientConnectedCheck="HH:MM:SS"
  comAuthenticationLevel="Default|None|Connect|Call|
                        Pkt|PktIntegrity|PktPrivacy"
  comImpersonationLevel="Default|Anonymous|Identify|
                        Impersonate|Delegate"
  maxWorkerThreads="num"
  maxIoThreads="num" />
```

securityPolicy

This element manages security policy:

```
<securityPolicy>
  <trustLevel name="value" policyFile="value" />
</securityPolicy>
```

sessionState

This element handles and directs session-state issues:

```
<sessionState mode="Off|Inproc|StateServer|SQLServer"
  cookieless="true|false"
  timeout="number of minutes"
  stateConnectionString="tcpip=server:port"
  sqlConnectionString="sql connection string" />
```

trace

This element configures the ASP.NET trace service:

```
<trace autoflush="true|false"
  indentSize="indent value"/>
```

trust

The trust element indicates the access security permission of an application:

```
<trust level="Full|High|Low|None" originUrl="url" />
```

webServices

This element controls the settings of XML Web services created using ASP.NET:

```
<webServices>
  <protocols>
    <add name="protocol name" />
  </protocols>
  <serviceDescriptionFormatExtensionTypes>
  </serviceDescriptionFormatExtensionTypes>
  <soapExtensionTypes>
    <add type="type" />
  </soapExtensionTypes>
  <soapExtensionReflectorTypes>
    <add type="type" />
  </soapExtensionReflectorTypes>
  <soapExtensionImporterTypes>
    <add type="type" />
  </soapExtensionImporterTypes>
  <wsdlHelpGenerator href="help generator file"/>
</webServices>
```

Deploying the Dreamweaver Control

Perhaps the most important thing you'll do when developing a .NET application in Dreamweaver MX will occur when you deploy the Dreamweaver Control. The Dreamweaver Control is a DLL developed by Macromedia specifically for use with Dreamweaver MX. This custom control is analogous to having hundreds of lines of code within a server script tag, right in your web page. The difference is that in the case of Dreamweaver Control, the code for the script lives in a different file (a source file called `DreamweaverCtrls.cs`, available in the Macromedia directory that is installed when you install the program, at `DreamweaverMX\Configuration\ServerBehaviors\Shared\ASP.Net\Scripts\Source`. This script was compiled "at the factory." Macromedia used the .NET compiler to compile the code into the DLL, which turned it into a binary executable that .NET can access anytime the control is called from within Dreamweaver.

Whenever you use Dreamweaver MX to create, for example, a DataSet (Insert > Application Objects > DataSet), you'll see that the Dreamweaver's code output contains special tags such as `MM:DataSet`. The definitions for these kinds of tags are made available through the `DreamweaverCtrls.dll`. We'll be covering the hows and whys of this methodology in Chapter 5.

When you create a .NET-driven data behavior, you'll need to deploy the Dreamweaver Control in order for .NET to recognize the Dreamweaver tags. To do this, go to Site > Advanced > Deploy Supporting Files. You'll see a dialog box that asks you how you access your site and to which directory you want to deploy (see Figure 1.10). Always deploy to the bin directory of your application. This is Dreamweaver's default. If the directory doesn't exist, Dreamweaver will create one for you.

FIGURE 1.10:

Deploying the
DreamweaverCtrls.dll



Wrapping Up

If you want to start building applications properly from the ground up, configuring Dreamweaver MX to work with .NET involves more than simply setting up your site definition.

Think of your website as a top-down process from the very beginning. Everything inherits from the top down. In other words, if the top level of your configuration indicates that your pages should be rendered in U.S. English, then all of them will be rendered in that language until ASP.NET encounters a directory in your site that overrides this settings. Moreover, this is how ASP.NET handles *all* object inheritance when code is involved, so it's a concept you'll want to remember as you dive into application development in future chapters.

There are two steps to take in order to begin your site:

1. Define a site.
2. Set up your IIS environment for production.

Technically, the additional configuration options are optional. You'll be able to serve pages without them, but your efficiency will be compromised—so we strongly recommend that you also configure the following as quickly as possible:

- Dreamweaver MX for use in versioning software
- The Web.config file, or at least the debugging elements it uses

The next chapter will look at the critical concept of workflow, examining how a web development team moves across a project from inception to final production using Dreamweaver MX.