Chapter No. 4
"Adding Security and Membership to a Content Management System"
In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.4 "Adding Security and Membership to a Content Management System"

A synopsis of the book’s content

Information on where to buy this book

About the Authors

Curt Christianson has been involved in the tech community since the mid 1990's and has been a professional developer for more than a decade. He is an active community contributor on the ASP.NET forums, as well as a Forum Moderator. He has won six Microsoft Most Valuable Professional (MVP) awards for his work with ASP/ASP.NET. He is writing a number of open source add-ins and starter kits. He is based in Wisconsin, U.S.A. as a professional developer, as well as contributing to books and articles, both printed and on the Internet.

Curt is in the process of entering into the life of a married man—thanks to his better half Jessyca. They plan on settling down with lots of little ones running around.

Jeff Cochran is a Senior Network Specialist for the City of Naples, Florida. A large part of his job includes web design and coding, as well as web server management. Jeff has nearly two decades of experience with the Internet, having started one of the first Internet Service Providers in Southwest Florida, and has worked with Windows and Unix-based web servers. Now primarily concentrating on Windows technologies, Jeff has been a Microsoft MVP for Microsoft's Internet Information Server for nearly a decade, and is active in the ASP Classic and ASP.NET communities as well.

Jeff has been married for twenty years to Zina, a graphic designer and, according to most accounts, the driving force that keeps him focused on… Oh look – A Pony! In the off-hours, Jeff and Zina spend much of their time remodeling a 1950's bungalow in Naples, Florida, trying to keep the rain out and the cats in. Jeff also has a long-term addiction to classic pinball machines, tropical fish, and off-road vehicles, all of which compete with home repairs for a share of his income.
ASP.NET 3.5 Content Management System Development

ASP.NET Content Management Systems are often at the heart of many businesses and customer interfaces. They help you to maintain and update content on a web site, even if you have little or no web design or programming experience. Imagine how great you'll feel when you have all the knowledge to get your site up and running quickly and also extend it into the future.

This book walks you through the creation of a functional Content Management System using the ASP.NET programming language. You will learn how to build your site in a number of ways, allowing customization. You can set up users and groups, create valuable content for your users, and manage the layout of your site efficiently when you have this book in hand.
What This Book Covers

Chapter 1 covers planning and building your first Content Management System.

Chapter 2 is about how to replace the file-based system with a database version. It also explores SqlDataSource, and using SQL Server 2005 Express as a source for data in our application.

Chapter 3 covers Content Management System architecture. It helps us build the database, a data access layer, a business logic layer, and a presentation layer for our Content Management System.

Chapter 4 discusses how to configure ASP.NET forms authentication, along with how to provide controls for users to log in, as well as ways to secure the content displayed on the pages.

Chapter 5 covers the basics of how to display your articles, how to create them, and how you may want to extend them.

Chapter 6 covers the concepts of why we lay out the site in a particular way, as well as beginning to help us understand all the pieces involved in this process.

Chapter 7 discusses a great deal about dynamically providing content to the users. It explores streaming files and images from the database, as well as generating RSS feeds "on the fly".

Chapter 8 covers maintaining users, adjusting permissions, approving Articles, and viewing site settings and stats—all key aspects of the Control Panel, which could be called the "brain" of any CMS.

Chapter 9 discusses a few additional options such as upsizing SQL server, using base pages and inheritance, and so on that may help extend a CMS.
Security is a concern in any web application, but the security this chapter deals with is that of user accounts, membership and roles. We'll be using the ASP.NET membership and roles functions to allow certain users such as administrators to perform specific tasks. These tasks may include managing the application, while other users such as content editors, may be restricted to the specific tasks we want them to manage such as adding or changing content. User account management can be handled either by the application (in our case, our Content Management System) or by Windows itself, using standard Windows authentication functions, as well as file and folder permissions.

The advantage of an application-based user authentication system is primarily in cost. To use Windows authentication, we need to purchase Client Access Licenses (CALs) for each user that will access our application. This is practical in an intranet, where users would have these licenses to perform other functions in the network. However, for an Internet application, with potentially thousands of users, licensing could be extremely expensive.

The drawback to an application-based system is that there is a lot more work to do in designing and using it. The Windows authentication process has been around for years, continually improved by Microsoft with each Windows release. It scales extremely well, and with Active Directory, can be extended to manage just about anything you can think of.
In this chapter, we will discuss:

- Membership—what it is and how it works
- Authentication—what it is and how to incorporate it into your application
- Setting up a basic application
- Creating the membership/authentication database pieces
- Adding a membership provider to the application
- Creating a login page and the controls associated with it
- Using the ASP.NET configuration tool and creating a login
- Forms authentication
- Membership roles

**ASP.NET membership**

Fortunately, Microsoft has provided relief for application-based authentication drawbacks in the 2.0 version of the ASP.NET framework, with the ASP.NET membership functions, and in our case, the SqlMembershipProvider. The membership API makes it simple for us to use forms authentication in our application, retrieving authentication and membership information from a membership provider. Similar to the classes we created in the last chapter for our data access layer and business logic layer, the membership provider abstracts the membership details from the membership storage source. Microsoft provides two providers—the ActiveDirectoryMembershipProvider that uses Active Directory and the SqlMembershipProvider that uses an SQL server database for the user data store.

By default, ASP.NET authentication uses cookies—small text files stored on the user's system—to maintain authentication status throughout the application. These cookies normally have an expiration time and date, which requires users to log in again after the cookie has expired. It is possible to use cookies to allow the client system to authenticate the application without a user login, commonly seen as a "Remember Me" checkbox in many web site login pages. There is naturally a downside to cookies in that a client system may not accept cookies. ASP.NET can encode the authentication information into the URL to bypass this restriction on cookies. Although in the case of our application, we will stick with the cookie method.
Forms authentication secures only ASP.NET pages. Unless you are using IIS7, and the integrated pipeline, where ASP.NET processes all file requests, the ASP.NET DLL won't be called for non-ASP.NET pages. This means that you cannot easily secure HTML pages, PDF files, or anything other than ASP.NET through forms authentication.

**Configuring and using forms authentication**

Let's start learning ASP.NET forms authentication by walking through a brand new application. We'll then add it to our Content Management System application. Forms authentication is actually quite simple, both in concept and execution, and a simple application can explain it better than adopting our current CMS application. Of course, we eventually need to integrate authentication into our CMS application, but this is also easier once you understand the principles and techniques we'll be using.

**Creating a new application**

Start by opening Visual Web Developer 2008 Express and creating a new web site by clicking on **File | New Web Site**. Use the **ASP.NET Website** template, choose **File System**, and name the folder **FormsDemo**.
When the site is created, you are presented with a Default.aspx page created with generic code. We will use this as our home page for the new site, although we need to modify it for our needs.

Creating the home page

Visual Web Developer 2008 Express creates a generic Default.aspx file whenever you create a new site. Unfortunately, the generic file is not what we want and will need modification. The first thing we want to do is make sure our site uses a Master Page, just as our Content Management System application will. To do this, we could delete the page, create our Master Page, and then add a new Default.aspx page that uses our Master Page. In the case of a brand new site, it's pretty easy, but what if you have developed an extensive site that you want to convert to Master Pages? You would want to add a Master Page to an existing site, so let's go ahead and do that.

Create the Master Page

We will create a Master Page just as we did in the previous chapter. Leave the Default.aspx file open and press Ctrl+Shift+A to add a new item to the solution. Choose the Master Page template and leave the name as MasterPage.Master. Place the code in a separate file and click Add to create the Master Page. You will notice that this creates the same generic code as in the previous chapter. Unfortunately, our Default.aspx file is not a content page and won't use the MasterPage.Master we just created, unless we tell it to.

To tell our Default.aspx page to use the MasterPage.Master, we need to add the MasterPageFile declaration, in the @ Page declaration, at the top of the file. Add the following code between the Language and AutoEventWireup declarations:

```xml
MasterPageFile="~/MasterPage.master"
```

This adds the Master Page to our Default.aspx page. However, content pages include only those Content controls that match the Master Page, not the full page code as our Default.aspx page currently does. To fix this, replace the remaining code outside the @ Page declaration with the following two Content controls:

```xml
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
  <h1>This is where the content goes.</h1>
</asp:Content>
```
We've left the Content1 control empty for the moment, and we've added a simple text statement to the Content2 control so that it can be tested. If you view the Default.aspx page in a browser, you should see the relatively uninteresting web page below:

![Image of a web page with a text statement](image)

This is where the content goes.

**Enabling forms authentication**

Okay, we have a boring home page for our new site. Let's leave it for a moment and enable forms authentication for the site, so we can restrict who can access our home page. The process of enabling forms authentication is simply adding a few lines to our web.config file. Or in the case of the generic web.config file, which we created while creating our new site, we simply need to alter a single line.

Open the web.config file in the new site and look for the line that says:

```xml
<authentication mode="Windows" />
```

Edit it to read:

```xml
<authentication mode="Forms" />
```

Save the web.config file and you have now enabled forms authentication for this site.

The default authentication mode for ASP.NET applications is Windows, which is fine if you're working in an intranet environment where every user probably has a Windows login for use in the corporate network anyway. Using Windows authentication, Windows itself handles all the security and authentication, and you can use the myriad of Windows utilities and functions such as Active Directory, to manage your users.
Adding Security and Membership to a Content Management System

On the other hand, with forms authentication, ASP.NET is expected to handle all the details of authentication and security. While ASP.NET 2.0 and later have sophisticated membership and profile capabilities (which we'll take advantage of later), there is no ASP.NET mechanism for protecting files and folders from direct access, outside of the application. You will still need to secure the physical server and operating system from outside of your application.

Creating the membership database

To use forms authentication and the SqlMembershipProvider, we need to create a database to authenticate against. This database will hold our user information, as well as membership information, so we can both authenticate the user and provide access based on membership in specific roles. For our demonstration, we will create a new database for this function, but later on we will incorporate the membership schema into our Content Management System database.

As we did in Chapter 2, we'll create a database with SQL Server Management Express, so open it and right-click Databases in the Object Explorer pane. Choose New Database and name it FormsDemo. Change the location of the database path to the App_Data folder of your FormsDemo web application—the default is C:\inetpub\FormsDemo\App_Data as shown below. Click OK and the new database will be created.
If you look at this database, you will see that it is empty. We haven't added any tables to it, and we haven't set up any fields in those non-existent tables. The database is pretty much useless at this stage. We need to create the database layout, or schema, to hold all the authentication and membership details. Fortunately, Microsoft provides a simple utility to accomplish this task for the 2.0 version of the ASP.NET framework – aspnet_regsql.exe. We'll use this too, in order to create the schema for us, and make our database ready for authentication and membership in our application.

To use aspnet_regsql.exe, we need to provide the SQL Server name and login information. This is the same information we set up SQL Server 2005 Express with in Chapter 2, and the same as shown in the login dialog when we open the database in SQL Server Management Studio Express, as shown below:

Note the server name, it will usually be {SystemName}/SQLEXPRESS, but it may be different depending on how you set it up. We used SQL Server Authentication with the sa account and a password of SimpleCMS when we set up SQL Server Express 2005, and that's what we'll use when we run the aspnet_regsql.exe tool.

To run aspnet_regsql.exe, you may browse to it in Windows Explorer, or enter the path into the Run dialog when you click on Start and then Run. The default path is C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet_regsql.exe. The utility may be run with command-line arguments, useful when scripting the tool or using it in a batch file, but simply running it with no parameters brings it up in a GUI mode. When the ASP.NET SQL Server Setup Wizard launches, click Next. Make sure that the Configure SQL Server for application services is selected and click on Next.
Adding Security and Membership to a Content Management System

The **ASP.NET SQL Server Setup Wizard** will ask for the server, authentication, and database. You should enter these according to the information from above.

![ASP.NET SQL Server Setup Wizard](image)

Click **Next** to confirm the settings. Click **Next** again to configure the database with the ASP.NET users and membership schema. Continue and exit the wizard, and the database is ready for us to use for authentication. If you were to open the FormsDemo database in SQL Server Management Studio Express, you would find that new tables, views, and stored procedures have been added to the database during this configuration process.

**Configuring the SqlMembershipProvider**

Our database is ready to use, but our application is not—at least not yet. We need to add a connection string in our `web.config` file so that we can connect to the database. We also need to add the `SqlMembershipProvider` information so that our application can access the database and use the new functions provided in our schema.

Open the `web.config` file in Visual Web Developer 2008 and find the default section that looks like:

```xml
<connectionStrings />
```
Replace it with:

```xml
<connectionStrings>
  <add name="FormsDemoConnectionString"
     connectionString="Data Source=.;SQLExpress;
     AttachDbFilename=C:\Inetpub\FormsDemo\App_Data\FormsDemo.mdf;
     Initial Catalog=FormsDemo.mdf;
     User ID=sa;
     Password=SimpleCMS"/>
</connectionStrings>
```

This will configure the database connection string so that we can use the database, as we did in Chapter 2.

To configure the SqlMembershipProvider, we need to add the AspNetSqlMembershipProvider to the Providers section of the Membership section, none of which we have in the default web.config. Immediately below the line that reads:

```xml
<authentication mode="Forms"/>
```

add the following code:

```xml
<membership defaultProvider="FormsDemoSqlMembershipProvider">
  <providers>
    <add name="FormsDemoSqlMembershipProvider"
         type="System.Web.Security.SqlMembershipProvider,
         System.Web, Version=2.0.0.0, Culture=neutral,
         PublicKeyToken=b03f5f7f11d50a3a"
         connectionStringName="FormsDemoSqlConnectionString"
         enablePasswordRetrieval="false"
         enablePasswordReset="true"
         requiresQuestionAndAnswer="true"
         requiresUniqueEmail="false"
         passwordFormat="Hashed"
         maxInvalidPasswordAttempts="5"
         minRequiredPasswordLength="7"
         minRequiredNonalphanumericCharacters="1"
         passwordAttemptWindow="10"
         passwordStrengthRegularExpression=""
    />
  </providers>
</membership>
```
Adding Security and Membership to a Content Management System

This provides the basic settings we need for our application. There are a few settings to take note of though:

- **defaultProvider**: We have designated a default provider for our application, as the `machine.config` file on our server uses `AspNetSqlMembershipProvider` as the default and expects a database named `aspnet.mdb` in the `App_Data` folder. Had we not created our own database and added the schema to it, `aspnet.mdb` would be the auto-created database. We do not want this for two reasons. The first is that every automatically configured application on the server would have the same database name. Also, it's easy to mix up database backups and maintenance schemes. More important though is that we have complete control and flexibility by creating our own database. The ASP.NET membership framework allows multiple providers so that we could split providers between databases for example. By specifically naming and creating our own database, and using it as the default for this application, we maintain explicit control.

- **applicationName**: We have set the `applicationName` to the root of the web application, which is what we want in this case. But this may not be where our application is located in a more complex application, and specifying the `applicationName` here would again provide us more explicit functionality. If we had not configured this, it would be set to the application root anyway. However, here we maintain control over it, as in the future, we may move the application.

- **enablePasswordRetrieval, enablePasswordReset, requiresQuestionAndAnswer**: These three are related, and set to the defaults. They determine whether a user can retrieve their password, reset their password, and whether or not answering a security question is required to perform either of those two functions. The default setting for these providers doesn't allow a user to retrieve his/her password because those would be sent to the user and could already be stolen by a hacker, but it allows a user to reset his/her password to a temporary one that can immediately be changed to the one known only by the user.

You also need to understand that these are defaults only in the `SqlMembershipProvider` we used, not the auto-created `AspNetSqlMembershipProvider`.

Password complexity in ASP.NET applications

ASP.NET password complexity often confuses both users and programmers. It is in the SqlMembershipProvider that the complexity is controlled. The default is a password with a minimum of seven characters, one of which must be non-alphanumeric, or not a number or letter. This means the password Password—which has eight characters, has both upper and lower case, and contains a zero—doesn’t meet the default requirements because it doesn’t contain a non-alphanumeric character. The password password! does meet the requirements, even though it has only lower case letters and no numbers. This is because the password has seven or more characters, and one of them, the exclamation point, is non-alphanumeric. You must decide on how complex you will require user passwords to be. More complex is more secure, but harder for users to deal with. At some point, security requirements become annoyances to the user and they will stop using your site. You may also use the passwordStrengthRegularExpression parameter to further refine your password strength, although the default is not to use it, leaving the expression blank. For example, the following code would require a password of at least seven characters, including one number and one non-alphanumeric character:

passwordStrengthRegularExpression="@\"(?=.*\d){1,}(?=.*\W){1,}""

You can find more about these, along with other SqlMembershipProvider properties, at http://msdn.microsoft.com/en-us/library/system.web.security.sqlmembershipprovider_properties.aspx.

Creating the login page

The first step to providing an authentication for users is creating a page for them to use to log into our application. The default name of this page for ASP.NET forms authentication is Login.aspx and we will stick to the defaults for this demonstration. So, start by adding a new item to our application in Visual Web Developer 2008 Express and choosing the Web Form template, naming it Login.aspx and selecting the MasterPage.master as your Master Page.

To add the login control to the page, enter the following code inside the Content2 ContentPlaceHolder control:

<asp:Login ID="Login1" runat="server">
</asp:Login>
If you save the Login.aspx file and run it in a web browser, you should see a page similar to this:

![Login Page Screenshot](image)

**Changing the default login page**

ASP.NET uses a default login page Login.aspx, and this is the URL that an unauthenticated user is redirected to when they try to access a page that requires authentication. To change this page name, we simply need to alter the authentication section of the web.config. The default web.config, and the one we used here, has a line similar to this:

```xml
<authentication mode="Forms" />
```

If we change this to:

```xml
<authentication mode="Forms">
    <forms loginUrl="UserLogin.aspx" />
</authentication>
```

Our application will then expect a page file named UserLogin.aspx and will use that as the login page for this application. We could also change the URL that logged in users are sent to if none is specified by using the `defaultUrl` parameter, similar to:

```xml
<authentication mode="Forms">
    <forms loginUrl="UserLogin.aspx"
        DefaultUrl="MembersPage.aspx" />
</authentication>
```

Although we have ignored these settings for this demo, good programming practices would include specifying these in the web.config for an application so that application doesn't accidentally inherit incorrect settings after deployment to a server with other applications on it.
Of course, if you try to log in using the Login.aspx page we just created, nothing will happen. We don't have a user account to log in with, so, let's create a quick one to test our logins.

**Creating a user account with the ASP.NET configuration tool**

Visual Web Developer 2008 Express has a built-in tool to help configure several different aspects of your application and IIS installation. We're going to use it to manage security by creating a new user account for accessing our web site. In Visual Web Developer, click on Website, and then ASP.NET Configuration. When the utility opens, click on the Security tab and you'll see a screen like this:

![Security Tab in Visual Web Developer](image)

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

Use the security Setup Wizard to configure security step by step.

---

Adding Security and Membership to a Content Management System

We have already created the database to store our user accounts in and we just need to create a user, so click on Create user and fill in the form on the following page, as shown below. Enter a User Name of User1, with a Password of Password!. Enter a valid email address, a Security Question of First Pet, and a pet's name such as Goldie. Click Create User, and after a couple of moments, you should get a confirmation that the user was created.

Windows authentication

In our application, we are using forms authentication to provide the security we need. We could use Windows authentication in a similar manner, for example in an intranet where users would normally already have Windows accounts. In Windows authentication, Windows users and groups take the place of user accounts and roles in forms authentication. You would create users and groups in Windows to be used to grant access to the application. Assigning user accounts to the groups would allow those users the access provided by group membership. Note that the Web Site Administration Tool cannot be used to manage users and groups in a Windows authentication application. You need to use the tools provided by Windows such as Active Directory. The advantage of Windows authentication is obvious—we have a single directory of users and access groups for all functions within your network. The disadvantage is the licensing costs of all those user accounts, if the only function they are needed for is to provide access to a single application.
Creating a login

Okay, we've created a user and we have a login page to log that user in. But why would a user log into our application? That's right, to reach pages or content that are restricted to logged in users. In our application, we will be restricting access to content based on whether a user has logged in or not. To do this, we make use of a LoginStatus control. This control will let us know the current status of the page viewer and also provide a way for that viewer to log into our application for further access.

Open the home page Default.aspx in Visual Web Developer, and locate the Content2 ContentPlaceHolder control. Immediately before the <h1> tag, enter the following code:

```html
<asp:LoginStatus ID="LoginStatus1" runat="server" />
```

That's it, just one line of code. Doesn't ASP.NET make this simple? When you save the file and run it in a browser, you should see a page like this:

![Login page]

Click on that little Login link and you'll see the Login.aspx page displayed, as that is the default login page for the ASP.NET login control. It will look similar to this:

![Login page]

Adding Security and Membership to a Content Management System

Enter a user name of **User1** and a password of **Password!**, as we used when creating our user account. You will then be authenticated and returned to the home page, where the login link has now become a **Logout** link, as shown below:

So, with a few lines of ASP.NET code, we have created an authentication system for our application. Of course, it's not really our application, just a demonstration, so let's move on and add these functions to our SimpleCMS application. We'll also need to extend this a bit more.

### Adding forms authentication to our CMS

Now that you understand the process behind forms authentication, we need to add it to our application. The process will be slightly different because we already have a database to use, but without the ASP.NET membership schema. We'll add that to the database and then create some user accounts and membership roles to handle the security for our application. We'll also secure some of our content and add a menu to our Master Page to navigate between the pages of our Content Management System.

### Preparing an existing SQL database

As we have an existing database, we can't create a new database for our membership and authentication system. Well, actually we could, but using a second database is problematic when we upload the application to a host because many web hosting companies allow only a single database under the hosting plan. Besides, we can easily add the membership schema the same way we did earlier in the chapter with our empty database, using `aspnet_regsql.exe`. Previously we used the wizard; this time we'll use the command line. If you take a look at the database in SQL Server Management Studio Express now, before we execute the command to add the schemas, you should see the few tables we created in earlier chapters, as shown in the following figure:
Chapter 4

The aspnet_regsql.exe tool

Using the command line, the executable is simple, as long as you know
the command-line arguments. The syntax and command arguments for
aspnet_regsql.exe are available online at http://msdn.microsoft.com/en-us/
library/x28wfk74.aspx. The following table shows the arguments we will use:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>What we use</th>
</tr>
</thead>
<tbody>
<tr>
<td>-S</td>
<td>The server name</td>
<td>\SQLEXPRESS</td>
</tr>
<tr>
<td>-U</td>
<td>The database username</td>
<td>sa</td>
</tr>
<tr>
<td>-P</td>
<td>The database password</td>
<td>SimpleCMS</td>
</tr>
<tr>
<td>-d</td>
<td>The database name</td>
<td>SimpleCMS_Database</td>
</tr>
<tr>
<td>-A</td>
<td>The schema functions to install</td>
<td>All functions</td>
</tr>
</tbody>
</table>

Our command line will look like this (all one line):

```
aspnet_regsql.exe -S .\SQLEXPRESS -U sa -P SimpleCMS -d SimpleCMS_Database -A all
```
To run the command line, go to **Start** | **Run** and enter `cmd` in the **Run** dialog box. Press **Enter** and you will be at a command prompt. Type `cd \ C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\` and press **Enter** again, and you will be in the correct folder to find `aspnet_regsql.exe`. Note that you may need to change the path if your ASP.NET framework files are in a different location. Type the command line above and press **Enter**, and you should see that the command completed successfully, with a dialog similar to that below:

Now that we have executed the `aspnet_regsql.exe` command line, if you look at the database tables in SQL Server Management Studio Express, you should see the added table for the users, membership, and roles we will use in our application.
User accounts

Earlier in the chapter, we created a single user account for accessing protected content. In a real-world environment, we would normally have many user accounts, way too many to add each account to each page we wanted to protect. Fortunately, the ASP.NET framework provides us with membership roles that we can place user accounts in, allowing us to define our access by role, not by user account. But first, we need some user accounts.

Let’s start by creating three accounts in our application—User1, User2, and Administrator. Open the SimpleCMS web site in Visual Web Developer 2008 Express. Use the downloadable code provided for Chapter 4 of this book, it has the web.config file modified similar to what we did when we walked through the forms authentication demo earlier in the chapter. Open the Web Site Administration Tool by clicking on Website and then ASP.NET Configuration.

If you click on the Security tab, you will see that we have no users configured for this application. As you did earlier in the chapter, click on Create User and create the three users with user names of User1, User2, and Administrator. Use Password! as the password for each, and provide a valid email address for each (they can have the same email for testing). Also, provide a question and answer such as Favorite Color? and Blue. You can use the same question and answer for all three accounts if you wish. Each user entry should look something like the following:

If you return to the Security tab, you will notice that we have three user accounts, but no roles for those accounts. Let’s add them next.
Membership roles

ASP.NET membership roles provide the ability to group many individual accounts into a single role to provide access to a resource such as a page or application. Changing access for an individual user then becomes a simple task of assigning them to or removing them from the appropriate role. A single user account can belong to multiple roles to provide extremely granular access to the application resources if your security demands are extensive.

To add roles to our application, we first need to enable roles. On the Security tab of the Web Site Administration Tool, under Roles, you should see a link to enable roles. Enabling roles consists of simply adding the following line to the web.config file in the system.web section:

```xml
<roleManager enabled="true" />
```

Similar to the membership provider we created earlier, roles require a role provider. We need to add this provider to the role manager, so edit the web.config roleManager section to read:

```xml
<roleManager enabled="true">
  <providers>
    <clear/>
    <add name="AspNetSqlRoleProvider"
      connectionStringName="SimpleCMS_DatabaseConnectionString"
      applicationName="/"
      Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
  </providers>
</roleManager>
```

This adds an AspNetSqlRoleProvider that uses our connection string to the SimpleCMS database. At this point we have no roles defined, so let's create a few. Open the Web Site Administration Tool. If it's already open, you may need to close and reopen it because we modified the web.config file to add the role provider.

Now, open the Security tab. In the Roles section, click on Create or manage roles.

Let's create an administration role first. We'll need it to secure areas to just administrative access. Simply enter Administrator, click on Add Role, and you'll see the new role in the list. Add roles for Author, Editor, and Registered User in the same manner. The roles list should look something like the following figure when you finish:
Adding users to roles

Once we have users and roles created, we need to assign users to roles. To do this, use the Security tab of the Web Site Administration Tool, under the Users section, to manage users. You’ll see a list of user accounts, in our case all three of them, along with the ability to edit the user, delete the user, and edit the user’s roles. Click on Edit roles next to the Administrator user and you’ll see a checkbox list of user roles this account can be added to. Any roles currently assigned to the user will be checked. As there are currently none, check the Administrator role, and the Administrator user will be immediately added to the Administrator role, as shown below:
Adding Security and Membership to a Content Management System

If you were to look at the database tables that hold the user accounts and roles, you would see something like this for the users:

<table>
<thead>
<tr>
<th>ApplicationID</th>
<th>UserID</th>
<th>UserName</th>
<th>LoweredUserName</th>
<th>Mobile</th>
<th>IsAnonymous</th>
<th>LastActivityDate</th>
</tr>
</thead>
</table>
| 659366a91c5c... | 3141b3b0-9e93-4632-b1b3-9e3b057b642 | Administrator | administrator | Null | False | 12/5/2009 2:11:...
| 63502b67-1c5c... | 824816db-7122... | Author | author | Null | False | 12/5/2009 3:37:...
| 63502b67-1c5c... | 7de59631-4eaf... | Editor | editor | Null | False | 12/5/2009 3:37:...
| 63502b67-1c5c... | 05511f2e-429a... | Registered User | registered user | Null | False | 12/5/2009 2:11:...

You'll note that both the users and the roles contain an ApplicationID that defines what application these users and roles belong to, and that each user or role is identified by a UserID or RoleID. These are automatically created by the ASP.NET membership framework and are globally unique identifiers (GUIDs), which ensure that the specific user or role is unique across all possible applications and uses of this specific database store.

You would also find in the database a table that identifies users in roles, looking something like this:

<table>
<thead>
<tr>
<th>UserID</th>
<th>RoleID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0e1b5f8f-af90-402c-9dca-302600e99fbc</td>
<td>3141b3b0-9e93-4632-b1b3-9e3b057b642</td>
</tr>
</tbody>
</table>

[124]

You'll notice that this is a joining table, used in a database when there is a many-to-many relationship. Many users can belong to a role and a user can belong to many roles, thus the use of this table. You'll also notice that the database table uses the UserID and RoleID, making it very hard to simply look at this table directly to find what users are assigned to what roles. Fortunately, with the ASP.NET framework, you're isolated from having to work directly with the database, as well as relieved from having to create it and the code needed to access it.

### Login page

We'll create the login page the same way we did with our demo application. Open the site in Visual Web Developer 2008 Express and add a new item to the application. Choose Web Form as the template and name it Login.aspx. Select the SimpleCMS.master as your Master Page and add the login code to the Content2 ContentPlaceHolder control as done before. Your login page should look very similar to our demo application.

### New user registration

Previously, we added user accounts to the database through the Web Site Administration Tool. This becomes impractical in our application for two reasons. The first reason is that the Web Site Administration Tool is not designed to work outside of the same system the site is hosted on. This makes using our application on a web host problematic. The second is that we really don't want to manually enter every user into the system, that's too much work. The ASP.NET framework makes life easy for us through the CreateUserWizard control, allowing users to add their own information to the user database and thus sign up for accounts on our system.

To add the CreateUserWizard to our login page, add the following code inside the Content2 ContentPlaceHolder control, immediately below the login control we added:

```xml
<asp:CreateUserWizard ID="CreateUserWizard1" runat="server">
   <WizardSteps>
      <asp:CreateUserWizardStep ID="CreateUserWizardStep1" runat="server">
      </asp:CreateUserWizardStep>
      <asp:CompleteWizardStep ID="CompleteWizardStep1" runat="server">
      </asp:CompleteWizardStep>
   </WizardSteps>
</asp:CreateUserWizard>
```
If you run the page in your browser, you should see something like:

![Login and CreateUserWizard controls](image)

You'll notice the same login control we used in our demonstration application, plus a new control that allows a user to sign up for an account. The CreateUserWizard control reads the Membership settings from our `web.config` file and populates the control accordingly. In our case, it asks for the user name, password, email, and both the security question and answer. This control also provides client side validation of the entries, requiring that each text box have an entry before submitting the form, and validating that the password entered meets the password requirements for our application.

Go ahead and sign up a new user, entering all the required fields and clicking on `Create User`. You should get a page similar to the one shown next, indicating that the user account has been successfully created.
Naturally, we want to create a more appropriate design for this page and these controls. It would help them look better and be more intuitive for users who want to register a new account, versus those who already have an account and wish to login. One of the simplest ways to do this is to open the **Design View** of the `login.aspx` page in Visual Web Developer 2008 Express, right-click on the `Login` control, and then choose **Autoformat**. Pick a format such as **Classic**, and your control will automatically take on that format. Doing the same with the `CreateNewUser` control should look similar to:
Adding Security and Membership to a Content Management System

If you open the code for the login.aspx page, you'll see the formatting for the controls has been added automatically. In a later chapter, we will work on formatting and layout options, along with the layout techniques. However, for now, let's get back to securing the content on our important pages.

Securing content

Okay, our application now has user accounts and roles for those users, but just how do we use them to secure the content in our Content Management System? In our demonstration, we secured entire pages and restricted access to those pages to specific accounts. But in our Content Management System, we want to secure the content itself, not the page. And if content is secure, we want to let our users know that they need to create an account and log in to see the content.

Let's begin by requiring users to have an active account to view an article from our database. Open the Default.aspx file in Visual Web Developer 2008 Express, and look at the FormView control that displays our article using the ArticlesBLL class, which in turn uses the DataSet1TableAdapters class. We don't want to change the functionality of that code, we just want that code to be available only to those users who have logged into our application. To do this, we'll use a LoginView control.

Change the FormView control section to the following code:

```xml
<asp:FormView ID="FormView1" runat="server">
    <ItemTemplate>
        <asp:LoginView ID="LoginView1" runat="server">
            <AnonymousTemplate>
                <p>We're sorry, this article requires you to have an account and be logged in to view the article.</p>
                <p><a href="login.aspx">Register or Login</a></p>
            </AnonymousTemplate>
            <LoggedInTemplate>
                <h2>
                    <asp:Label ID="Label1" runat="server"
                        Text='<%# Bind("ArticleName") %>'>
                </h2>
                <hr />
                <asp:Label ID="Label2" runat="server"
                    Text='<%# Bind("Article") %>'>
                </asp:Label>
                <hr />
            </LoggedInTemplate>
        </asp:LoginView>
    </ItemTemplate>
</asp:FormView>
```

[ 128 ]
Chapter 4

The **LoginView** control shown here has two templates—an **AnonymousTemplate** and a **LoggedInTemplate**. These do just what they indicate, provide the user with the information that is laid out in the appropriate template, either Anonymous orLoggedIn Template, based on their current login status. If you run the page in your browser, you should see the following:

![Image of LoginView control](image)

If you then click on the **Register or Login** link, and log in as a registered user, you should see the **Default.aspx** page, complete with the article from the database.

![Image of Default.aspx page](image)
Login status
There is an even more elegant way to handle login requirements in the
ASP.NET 2.0 framework via the LoginStatus control, similar to what we did in our
demo application. We can use it to add a login link to every page, so we don't have
to build a login link into all the LoginView controls we might add to our application.
This control displays a login or logout link, according to the logged in status of a
user. This means if a user is not logged in, we will automatically show them a link to
do so. That link will take them to the login page we created earlier.

Open the SimpleCMS.master Master Page file in Visual Web Developer 2008
Express. At the bottom of the page, you will find the copyright statement we added
earlier. Immediately below that line, add this code of the LoginStatus control:

<asp:LoginStatus ID="LoginStatus1" runat="server" />

That's it, everything we need to add a login link on every page in our application. We
can go back and delete the line from our Default.aspx LoginView that reads:

<p><a href="login.aspx">Register or Login</a></p>

If you then view the Default.aspx page in a browser, it should look like the
following figure when you are not logged in:
The Login link is automatically displayed on any page where the user is not logged in because it is part of our Master Page. If a user is already logged in, the link simply changes to a Logout link.

Password recovery

A major headache with almost any web site on the Internet that requires registration is that you often do not want to, or even cannot, use the same password as you do on other sites. This results in most users having multiple passwords, and most users forgetting at least some of those passwords. The ASP.NET 2.0 framework has a PasswordRecovery control for just this purpose. Let's go ahead and add it to our application.

In Visual Web Developer 2008 Express, add a new web form with the name ForgotPassword.aspx and then select the SimpleCMS.master page Master Page file. In the Content2 ContentPlaceholder control, add the following code:

```html
<asp:PasswordRecovery ID="PasswordRecovery1" Runat="server">
</asp:PasswordRecovery>
```

Open the Design View for this page, and AutoFormat the control to the same Classic format we used in the other login controls.

To link to this page, we'll use a LinkButton control on the login.aspx page. Open the page and add this code after the Login control:

```html
<asp:LinkButton ID="LinkButton1" runat="server" PostBackUrl="/ForgotPassword.aspx">
Forgot Password?
</asp:LinkButton>
```
Adding Security and Membership to a Content Management System

Save these files and when you run the login.aspx page in the browser, you should see the Forgot Password? link below the login control. Clicking on that link will show our ForgotPassword.aspx page, which looks like this:

A user entering their login name will then be presented with their challenge question and must answer it to receive their password. A correct answer results in the user receiving an email, containing his/her password, to his/her account. At this point, you will receive an error if you try to recover your password because we have not set up any email capability in our application. We'll take care of that in a later chapter.
Summary

In this chapter, you learned how to configure ASP.NET forms authentication, along with how to provide controls for users to log in, as well as ways to secure the content displayed on the pages. We used the \texttt{aspnet_regsql.exe} utility to create the database for membership and authentication. We also used the ASP.NET Configuration utility to configure some authentication parameters for our web application, add users and roles, and assign users to roles. We also created pages that were secured from access by unauthorized users.

When we added these features to our application, we expanded our login page to allow users to register a new account and even to recover a password if they forgot it. We used the Login and CreateNewUser controls, which are built into the ASP.NET 2.0 framework, and we used the AutoFormat option to format these controls as the user will see them. We also used the LoginView control to restrict access to an article on our page, as well as the LoginStatus control to add a login link to all of our pages through our Master Page.

If you are interested in more depth on the ASP.NET membership controls, you should check out the MSDN Patterns and Practices information at \url{http://msdn.microsoft.com/en-us/library/ms998347.aspx}. You will also find more information in the online tutorials at \url{http://www.asp.net/learn/security/}.

As we move through future chapters, we'll add a few more features to our application related to users and user management. In Chapter 9, we'll build a control panel that allows us to manage user accounts and role memberships without using the ASP.NET Configuration utility. We'll also do some advanced formatting of our pages and controls in Chapter 8 and work more with Master Pages in Chapter 6.

In the next chapter, we'll build our first complete module for our application, an Articles module that will allow us to create and manage the articles in our database. This module is the basis for our dynamic content in the Content Management system, although we'll deal with static content such as existing documents, pictures, and other files in a later chapter.
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.