CHAPTER 1

Getting Started

Performance is always an issue. This book will show you how to optimize ASP.NET 2.0 application access to SQL Server databases. You can leverage the close integration of ASP.NET 2.0 and SQL Server to achieve levels of performance not possible with other technologies. You'll investigate in detail the middle ground between ASP.NET 2.0 and SQL Server and how to exploit it.

This book demonstrates all concepts with professional code, so the first thing you need to do is set up your development environment. I'll cover related issues along the way.

This chapter covers the following:

- Preparing your environment
- Managing provider services
- Configuring providers
- Creating users and roles

Preparing Your Environment

I typically run several Microsoft Virtual PC environments and move between them as my needs change. When I first set up my initial virtual environments, I create as many as I need. One will be my main development environment. This has been helpful as I keep a backup image of the initial environment immediately after I create it. As I try out beta releases or third-party add-ons, I may begin to dirty up my environment. Because the uninstall process may not completely clean up Windows installations, the virtual environments come in handy.

For example, at a local user group session, a presenter was demonstrating a tool that integrated with Visual Studio. Instead of installing the tool in my current development environment, I cloned a fresh environment and used it to try the tool. Afterward, I turned off the environment and deleted it to clear up space for another environment. My primary environment wasn't affected at all.

I also create two drives within my virtual environment. The `C:` drive holds the operating system, while the `D:` drive holds my data. These drives are represented by virtual hard drives. And if I decide to clone my data drive, I can use it with another system as I see it. Because I typically develop every project by using source control, I can start with a fresh environment, pull down the current project versions, and start development again.

To further leverage virtualization, you can also create a virtual hard-drive installation of your development environment and set the files to read-only. Then you can create a new virtual image that uses the read-only copy as a base image while keeping all changes on the

secondary image. When you are given the option to create a new virtual hard drive, you can choose the Differencing option, as shown in Figure 1-1.

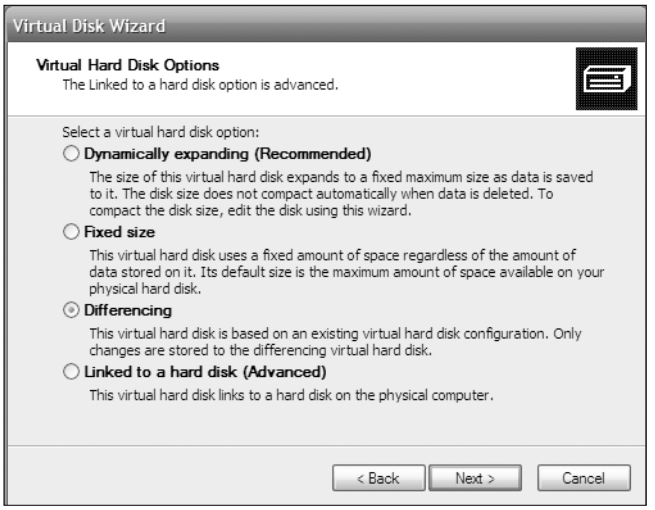

Figure 1-1. *Creating a differencing virtual hard disk*

The differencing image allows you to leave your base image untouched as you begin to use a fresh environment for development. This conserves space on a hard drive, which can fill up quickly; a typical virtual image can grow to 20 GB, not including the data image. And because you can have multiple virtual images using the same base image, your environment can quickly be prepared for your work as you need it.

One time my coworker needed to start work on a short .NET 1.1 project the next day. She had only Visual Studio 2005 installed and so was facing a long install process that typically takes several hours to complete. But because I had already prepared virtual images for .NET 1.1 and .NET 2.0 development, I had her ready to go in under 20 minutes. The longest wait was for copying the image off the file server. After she was finished with the short project, she was able to remove the temporary environment.

Having access to these prebuilt environments has been a major time-saver. I can experiment with tools and techniques that I would not have a chance to try if I had to cope with the consequences of having alpha- and beta-release software mixed in with my main development environment.

Project Organization

For each environment, I place all of my projects into `D:\Projects`. And I explicitly have the solutions at the root of each project. I have seen how some teams create solution files only as a side effect of creating projects, but that undercuts the advantages you get when you properly manage your projects within a solution.

For a typical web project, I start with the ASP.NET website in a blank solution. I then add a class library and call it `ClassLibrary`. I put as much of the code for the website into this class library, for reasons I'll cover later. Then I associate the class library to the website as a reference, which the solution records as a dependency. This is quite helpful, as the new ASP.NET 2.0

website project model does not include a project file that maintains a manifest for files and dependencies. I add a database project called `Database`, which holds all of my database scripts for creating tables and stored procedures, and scripts to prepare the database with supporting data. (Database projects require Visual Studio 2005 Professional Edition.) Finally, I create a solution folder called `Solution Items` and add a text file named `README.txt` that provides the basic information for the project, such as the name, description, requirements, dependencies, and build and deployment instructions.

The result of all this is the solution structure shown in Figure 1-2.

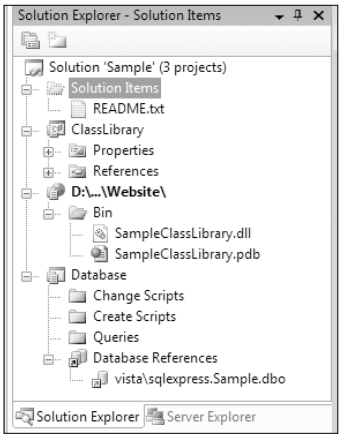

Figure 1-2. *Typical solution environment*

When you first set up a blank solution and add your first project or website to it, you may find that the solution goes away. This is a default setting for Visual Studio, which you can change. From the Tools menu, choose Options and then click the Projects and Solutions item, as shown in Figure 1-3.

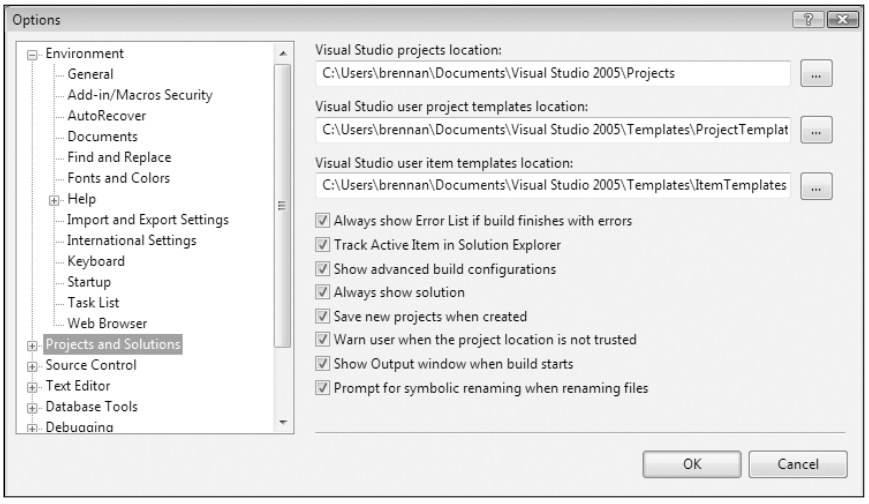

Figure 1-3. *Setting the Always Show Solution option*

This sample solution will be the template for all the examples in this book. Using the same basic structure for all projects provides a consistent basis for build automation. It also keeps everything in the same place for every application you develop. Being able to easily search the solution for a reference to a database table and then get the table creation script and any stored procedures using that table is very convenient. Most developers simply write and keep their table Data Definition Language (DDL) and stored procedures in the database and move them around by using tools within the database, never saving them as scripts that can be version-controlled in a Visual Studio project. This common practice fails to leverage one of the great strengths of Visual Studio 2005 and makes it harder to re-create database objects from scratch. As a result, changes to improve the application when it comes to database updates are avoided because of all the extra work necessary to make the change. When your environment allows you to work in a very agile way, you can take on tasks that may not be attempted otherwise.

Common Folders

As you work on many projects, you'll accumulate tools, templates, and scripts that are useful across multiple projects. It's helpful to place them into a common folder that you manage with source control so that the developers among multiple teams can leverage them as they do their work. With the Solution layout in Figure 1-2, it's trivial to drop in an MSBuild script and `CMD` scripts to a new project to provide build automation. You'll use the following folders for tools, templates, and scripts throughout the book (see Figure 1-4):

```
D:\Projects\Common\Tools

D:\Projects\Common\Templates

D:\Projects\Common\Scripts
```

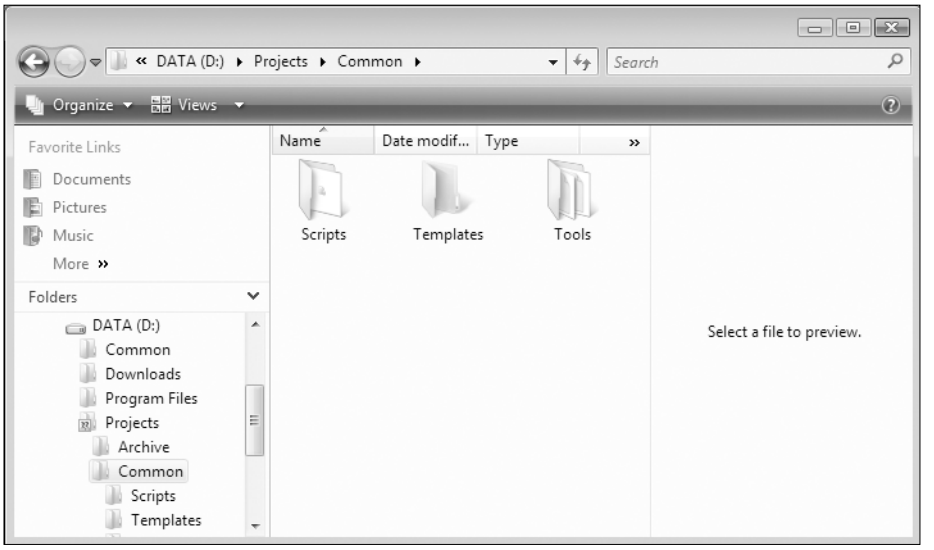

Figure 1-4. *Folders for tools, scripts, and templates*

To make it easier to script against these folders, let's add some system environment variables for these locations (see Figure 1-5):

```
DevTools = D:\Projects\Common\Tools

DevTemplates = D:\Projects\Common\Templates

DevScripts = D:\Projects\Common\Scripts
```

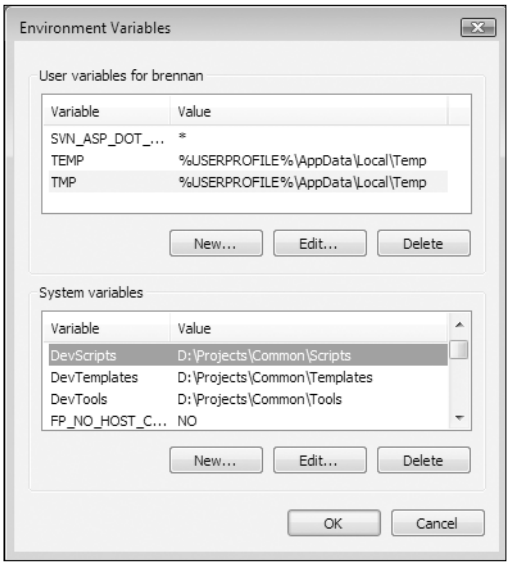

Figure 1-5. *Environment variables for development*

You'll build on these folders and variables throughout the book to enhance your common development environment.

Datasource Configuration

The datasource is the mechanism used to connect an ASP.NET 2.0 web application to data. For an ASP.NET application, the data is typically in a SQL Server database. To connect to a datasource, you use a connection string that sets various options for connecting to a database. There are many options available with connection strings. For the most basic connection string, you need the location of the database and the authentication details to access the database. The following is an example of such a connection string:

```
server=localhost;database=Products;uid=webuser;pwd=webpw
```

Notice that this includes `server`, `database`, `uid`, and `pwd` parameters, which provide all that is necessary to access the `Products` database on the local machine. This form should be very familiar to an ASP.NET developer. However, there are alternatives I will explain shortly.

MIXED-MODE AUTHENTICATION

With SQL Server, you can choose to allow for Windows authentication, SQL Server authentication, or both. This choice is presented to you when SQL Server is installed. SQL Server accounts are useful when the database is hosted on a remote server without access to the Windows domain. Windows authentication is helpful during development, when your users and the database server are running on the domain. For development and staging databases, you may grant different levels of access to groups such as database developers and web developers and then place people in those groups accordingly. For the production databases, you use a user account that is not shared with the entire development team so that only those limited users who should have access, do. The applications working with these databases just need the connection string updated to change authentication modes.

In an ASP.NET application, datasources are configured in the `Web.config` file in a section named `connectionStrings`. Each connection string is added to this section with a name, connection string, and provider name—for example:

```
<connectionStrings>
  <add name="SampleDatabase" connectionString="..."
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

In a team environment, you'll likely use a source-control system. You'll place your `Web.config` file into source control so that each member of the team has the same configuration. However, sharing a source-controlled configuration file can present some problems.

For starters, a common set of authentication values may be used. Using a Windows user account (for example, of one of the team members) exposes the password to the whole team—which is bad practice, especially if the password policy requires routine updates. Another approach is to use a SQL Server account that is shared by the team. This way is better than using a shared Windows account, but the following option works best. Each project should be configured with a trusted connection string so that the current user's account is used to connect with the database—for example:

```
server=localhost;Trusted_Connection=yes;database=Products;
```

Instead of providing the `uid` and `pwd` parameters, this uses Windows authentication to access the database. All developers use the same connection string but individually access the database by using their own accounts.

A trusted connection allows you to control who has access to what in the databases. When there is a universally shared user account, everyone will have access to everything, and no individual role management is possible. This way, you can set up the web development team with access to create and modify stored procedures while not giving them access to directly alter tables or data. Meanwhile, you can have your database team manage changes to the tables and stored procedures. And if you do have business analysts or management accessing the database, you can give them enough access to do what they need to do, but nothing that would cause trouble with the databases.

A side benefit is that your web team can focus on their concerns without having responsibility for what is happening to the database internals. It also gives the database team the

freedom to change the database structure while using updated stored procedures to provide the same public interface for the web team, taking in the same parameters and returning the same result columns. Keep in mind that sometimes some team members will wear hats for the web and database teams. That is perfectly fine. Some senior members of the team may have the right level of mastery on every part of the system to handle those responsibilities. But not every team member will be ready to take on that level of responsibility. Isolating the latter group of developers to what they can manage will give them and the project leader some peace of mind knowing that the project is in the right hands.

Code and Database Separation

On one recent project, I took care of changes to the database while developing the new website from the ground up. The catch was that this website was being built as a front end for multiple online stores and had a unique database structure for each store. That presented a real challenge. I planned to build just one basket and order management component but had to work with different databases while minimizing the effort to integrate the data with various front ends.

To make the same front end work with these multiple back ends, I created a clean integration point by using a set of stored procedures that had the same name and a set of parameters for each of the databases. These stored procedures each returned the same columns or output parameters. Internally, each would join differently named tables and gather column data from different locations than the next database, but ultimately would return data that the front end could easily interpret and display to customers. To run this online store with a different database, all that was needed was to implement those stored procedures. No coding changes were needed for the websites.

Another side benefit was that my team member was proficient with T-SQL and intimately familiar with each of the databases. By isolating her focus in that space, she was very productive and able to complete the integrations for a new website rollout much more quickly that I could have. As she worked on those changes, I was able to focus on requested changes to the front end as well as performance optimization.

As an alternative to using a set of stored procedures as a reliable integration layer, you could consider a set of web services (Windows Communication Foundation, or WCF) as a service-oriented architecture (SOA). I am sure that could work, but when a database already allows a wide range of platforms to talk with it, you already have a cross-platform approach. It may not use Extensible Markup Language (XML) or Web Services Interoperability Organization (WSI) specifications, but SQL Server does work with .NET, PHP, Java, Delphi, and a whole range of languages and platforms over an Open Database Connectivity (ODBC) connection. And the developers using those languages and platforms will already have some basic proficiency with SQL, so they can jump right into this approach without touching a single line of C# or Visual Basic (VB).

Managing Provider Services

The provider model was introduced to ASP.NET 2.0. It is to web applications what plug-ins are to web browsers. Included with the ASP.NET providers are the Membership, Roles, and Profile providers. Each of these providers has a SQL Server implementation, among others. By default, the SQL Server implementations are preconfigured for an ASP.NET website. However,

you must support these implementations with resources in the database. To prepare these resources, you will use the `aspnet_regsql.exe` utility.

The Visual Studio 2005 command prompt provides easy access to this utility from the console. The utility itself is located in the .NET 2.0 system directory, which is included on that path defined for the special command line used by the Visual Studio 2005 command prompt. The utility can be run with no arguments to start the wizard mode, which is a visual mode. Despite the name, the wizard mode is not nearly as powerful as the command line, which offers more options.

For starters, the visual mode turns on support for all providers. To enable just the Profile or Membership provider, you can request just those features from the command line and not add support for the Roles provider (which you may implement with a custom provider or not at all). Get a full list of the available options with the following command:

```
aspnet_regsql.exe -?
```

The available services supported by the utility include Membership, Role manager, Profiles, Personalization, and SQL Web event provider. Each can be selectively registered with the database.

Using the Command Line

Most of the time that I work with a website using the available providers, I use only the Membership, Roles, and Personalization support and leave the others off. To add just the desired services, use the following `Add Provider Services.cmd` script in Listing 1-1.

Listing 1-1. *Add Provider Services.cmd*

```
@echo off
set REGSQL="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_regsql.exe"
set DSN="Data Source=.\SQLEXPRESS;Initial Catalog=Chapter01; ➥
Integrated Security=True"
%REGSQL% -C %DSN% -A mrpc
pause
```

This script can be placed at the root of a project, alongside the solution file, to make it easily available during development. It sets the location of first the utility and then the data-source to be used to host these features. Finally, the command is executed to add support for the Membership and Roles provider in the database.

The `-C` switch specifies the connection string, while `-A` specifies the list of services you want added to the database. To add all of them, you can simply specify `all` instead. The script takes a short time to complete. When it is done, you can see there will be new tables and stored procedures in the selected database.

Adding only the services that are going to be used is a practice of minimalism. Features have a tendency to be used if they are available, so by withholding the services that you do not plan to use, you save yourself the trouble of watching over them.

While you are developing a website with these features, you will occasionally want to reset everything and start from scratch. To do so, you can use a script to remove the provider services. However, the utility does not let you remove these services if there is data in tables

created by the utility. To help it out, you must delete the data in the right order because of the foreign key constraints among the tables.

To do so, use the following script named `WipeProviderData.sql` in Listing 1-2.

Listing 1-2. *WipeProviderData.sql*

```
--
-- WipeProviderData.sql
-- Wipes data from the provider services table so the services can be removed
-- (and added back fresh)
-- SELECT name FROM sysobjects WHERE type = 'U' and name like 'aspnet_%'
--

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND
  name = 'aspnet_WebEvent_Events')
  BEGIN
    delete from dbo.aspnet_WebEvent_Events
    END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND
  name = 'aspnet_PersonalizationAllUsers')
  BEGIN
    delete from dbo.aspnet_PersonalizationAllUsers
  END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND
  name = 'aspnet_PersonalizationPerUser')
  BEGIN
    delete from dbo.aspnet_PersonalizationPerUser
  END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND
  name = 'aspnet_Membership')
  BEGIN
    delete from dbo.aspnet_Membership
  END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND name = 'aspnet_Profile')
  BEGIN
    delete from dbo.aspnet_Profile
  END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND
```

```
  name = 'aspnet_UsersInRoles')
  BEGIN
    delete from dbo.aspnet_UsersInRoles
  END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND name = 'aspnet_Users')
  BEGIN
    delete from dbo.aspnet_Users
  END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND name = 'aspnet_Roles')
  BEGIN
    delete from dbo.aspnet_Roles
  END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND name = 'aspnet_Paths')
  BEGIN
    delete from dbo.aspnet_Paths
  END
GO

IF EXISTS (SELECT * FROM sysobjects WHERE type = 'U' AND
  name = 'aspnet_Applications')
  BEGIN
    delete from dbo.aspnet_Applications
  END
GO
```

You can place this script into your scripts folder, `D:\Projects\Common\Scripts`, for use in multiple projects. When you want to use it, load it into SQL Server Management Studio and run it in the context of the database you want wiped. Then you can run the removal script, `Remove Provider Services.cmd`, shown in Listing 1-3.

Listing 1-3. *Remove Provider Services.cmd*

```
@echo off
set REGSQL="%windir%\Microsoft.NET\Framework\v2.0.50727\aspnet_regsql.exe"
set DSN="Data Source=.\SQLEXPRESS;Initial Catalog=Chapter01; ➥
Integrated Security=True"
%REGSQL% -C %DSN% -R mrpc
Pause
```

This script is identical to `Add Provider Services.cmd`, except for the simple change in the command-line switch from `-A` to `-R`, which specifies that the services are to be removed. After the data is wiped, this script will run successfully.

Mixing and Matching Providers

Because of the nature of the providers, it is possible to deploy these provider services to either the same database that the rest of the website is using or an entirely different database. It is possible to even deploy the services for each individual provider to a separate database. There can be good reason to do so. In one instance, you may be running a website connected to a massive database that needs to be taken offline occasionally for maintenance. In doing so, it may not be completely necessary to take the provider services offline as well. You may also find that you get better performance by having your provider services hosted on a database on a different piece of hardware.

Your configuration options go much further, as explained in the next section.

Configuring Providers

When you first create a new ASP.NET 2.0 website with Visual Studio 2005, it is already preconfigured to work with a set of defaults. These defaults are set in the `Machine.config` file, which is a part of the .NET 2.0 installation. Typically it is located at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG`, or wherever .NET 2.0 has been installed onto your computer. The defaults for the provider configuration are near the bottom, in the `system.web` section, as shown in Listing 1-4.

Listing 1-4. *system.web in Machine.config*

```
<system.web>
  <processModel autoConfig="true"/>

  <httpHandlers />

  <membership>
    <providers>
      <add name="AspNetSqlMembershipProvider"
        type="System.Web.Security.SqlMembershipProvider, ➥
System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
        connectionStringName="LocalSqlServer"
        enablePasswordRetrieval="false"
        enablePasswordReset="true"
        requiresQuestionAndAnswer="true"
        applicationName="/"
        requiresUniqueEmail="false"
        passwordFormat="Hashed"
        maxInvalidPasswordAttempts="5"
        minRequiredPasswordLength="7"
        minRequiredNonalphanumericCharacters="1"
        passwordAttemptWindow="10"
        passwordStrengthRegularExpression="" />
    </providers>
  </membership>
```

```
  <profile>
    <providers>
      <add name="AspNetSqlProfileProvider"
connectionStringName="LocalSqlServer" applicationName="/"
        type="System.Web.Profile.SqlProfileProvider, System.Web, ➥
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    </providers>
  </profile>

  <roleManager>
    <providers>
      <add name="AspNetSqlRoleProvider"
connectionStringName="LocalSqlServer" applicationName="/"
        type="System.Web.Security.SqlRoleProvider, System.Web, ➥
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
      <add name="AspNetWindowsTokenRoleProvider" applicationName="/"
        type="System.Web.Security.WindowsTokenRoleProvider, XXX
System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    </providers>
  </roleManager>
</system.web>
```

A default connection string named `LocalSqlServer` is also defined, which looks for a file called `aspnetdb.mdf` in the `App_Data` folder (see Listing 1-5).

Listing 1-5. *connectionStrings in Machine.config*

```
<connectionStrings>
               <add name="LocalSqlServer"
connectionString="data source=.\SQLEXPRESS;Integrated ➥
Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User Instance=true" ➥
providerName="System.Data.SqlClient"/>
</connectionStrings>
```

The default `LocalSqlServer` connection is referenced by the Membership, Roles, and Profile provider configurations in the `system.web` block. The datasource specifies that this database exists in the `DATA_DIRECTORY`. That `DATA_DIRECTORY` for an ASP.NET 2.0 website is the `App_Data` folder. If you were to create a new website and start using the Membership and Profile services, this SQL Express database would be created for you in the `App_Data` folder. However, if you have the Standard or Professional Edition of SQL Server installed, this process would fail because SQL Express is required. This default configuration can be quite handy, but also dangerous if you do not adjust it for the deployed environment.

For each provider configuration, the parent block has multiple attributes for the respective provider implementation. And within that block, you have the ability to add, remove, or even clear the provider implementations. In your new website's `Web.config` file, you will want to clear the defaults set by `Machine.config` and customize them specifically for your needs.

Next you will configure a new website to use the SQL implementations of the Membership, Roles, and Profile providers. Before those are configured, you must prepare the datasource.

Assuming you have a sample website that will work with a database called `Sample` and that has been prepared with the provider services as described in the previous section, use the configuration in Listing 1-6.

Listing 1-6. *Custom Web.config*

```
  <connectionStrings>
    <add name="sampledb"
       connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=Sample; ➥
Integrated Security=True"
       providerName="System.Data.SqlClient"/>
  </connectionStrings>
```

During the development process, I use a trusted database connection and talk to the local machine. Now you are ready to configure the providers with the SQL implementations. And because the defaults in `Machine.config` already use the SQL implementations, let's start with them and then make our adjustments.

Membership Configuration

For starters, you'll organize the configuration block to make it more readable. You will be looking at this quite a bit during development, so making it easy to read at a glance will save you time. Place each attribute on a separate line and indent the attributes to line them up. Then make the `name`, `applicationName`, and `connectionStringName` the first few attributes. These are the critical values.

Next you want to ensure that this is the only Membership provider for this website. Add the `clear` element before the `add` element in the `membership` block to tell the ASP.NET runtime to clear all preconfigured settings. Then add an attribute called `defaultProvider` to the `membership` element and set it to the same value as the name for the newly added provider configuration. Listing 1-7 shows the Membership configuration. Table 1-1 covers the various settings that are available.

Listing 1-7. *Membership Configuration*

```
<membership defaultProvider="Chapter01SqlMembershipProvider">
  <providers>
  <clear/>
  <add
      name="Chapter01SqlMembershipProvider"
      applicationName="/chapter01"
      connectionStringName="chapter01db"
      enablePasswordRetrieval="true"
      enablePasswordReset="true"
      requiresQuestionAndAnswer="true"
      requiresUniqueEmail="false"
      passwordFormat="Clear"
      maxInvalidPasswordAttempts="5"
      minRequiredPasswordLength="7"
```

```
      minRequiredNonalphanumericCharacters="0"
      passwordAttemptWindow="10"
      passwordStrengthRegularExpression=""
      type="System.Web.Security.SqlMembershipProvider, ➥
System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
    />
  </providers>
</membership>
```

Table 1-1. *Membership Configuration Settings*

Setting	Description
`name`	Specifies the configuration name referenced by the `membership` element
`applicationName`	Defines the application name used as a scope in the `Membership` database
`connectionStringName`	Specifies the connection string to use for this provider
`enablePasswordRetrieval`	Specifies whether this provider allows for password retrieval
`enablePasswordReset`	Specifies whether this provider can reset the user password (enabled = `true`)
`requiresQuestionAndAnswer`	Specifies whether the question and answer are required for password reset and retrieval
`requiresUniqueEmail`	Specifies whether this provider requires a unique e-mail address per user
`passwordFormat`	Specifies the password format, such as cleared, hashed (default), and encrypted
`maxInvalidPasswordAttempts`	Specifies how many failed login attempts are allowed before the user account is locked
`minRequiredPasswordLength`	Specifies the minimal number of characters required for the password
`minRequiredNonalphanumericCharacters`	Specifies the number of special characters that must be present for the password
`passwordAttemptWindow`	The duration in minutes when failed attempts are tracked
`passwordStrengthRegularExpression`	A regular expression used to check a password string for the required password strength

Roles Configuration

Now you'll do much of the same with the Roles provider. This configuration block is called `roleManager`, and in addition to the `defaultProvider` attribute, it also has an attribute called `enabled`, which allows you to turn it on and off. Later you can easily access the `enabled` setting in your code by using the `Roles.Enabled` property. By default this value is set to `false`, so it must be set to add support for roles even if you have configured a provider. See the following roles configuration in Listing 1-8.

Listing 1-8. *Roles Configuration*

```
<roleManager defaultProvider="Chapter01SqlRoleProvider" enabled="true">
  <providers>
  <clear/>
  <add
    name="Chapter01SqlRoleProvider"
    connectionStringName="chapter01db"
    applicationName="/chapter01"
    type="System.Web.Security.SqlRoleProvider, ➥
System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
    />
  </providers>
</roleManager>
```

Profile Configuration

Finally, you can add support for the Profile provider. And as with the preceding two provider configurations, you will set the `defaultProvider` attribute to match the newly added configuration and also add the `clear` element to ensure that this is the only configured provider. The unique feature for the Profile provider configuration is the ability to define custom properties. In the sample configuration shown in Listing 1-9, a few custom properties have been defined: `FirstName`, `LastName`, and `BirthDate`. I will explain these properties in a bit; Table 1-2 lists the profile configuration settings.

Listing 1-9. *Profile Configuration*

```
<profile defaultProvider="Chapter01SqlProfileProvider"
  automaticSaveEnabled="true" enabled="true">
  <providers>
  <clear/>
  <add
     name="Chapter01SqlProfileProvider"
     applicationName="/chapter01"
     connectionStringName="chapter01db"
     type="System.Web.Profile.SqlProfileProvider, ➥
System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>
  </providers>
  <properties>
  <add name="FirstName" type="String" allowAnonymous="true" />
  <add name="LastName" type="String" allowAnonymous="true" />
  <add name="BirthDate" type="DateTime" allowAnonymous="true" />
  </properties>
</profile>
```

Table 1-2. *Profile Configuration Settings*

Setting	Description
`name`	Specifies the configuration name referenced by the `profile` element
`applicationName`	Defines the application name used as a scope in the `Profile` database
`connectionStringName`	Specifies the connection string to use for this provider

You can see those custom properties near the end. The Profile provider allows you to manage properties that are bound to the ASP.NET user. You can define anything that can be serialized. In the three examples, both `String` and `DateTime` objects can easily be serialized for storage in the SQL Server database. However, the properties can be much more complicated objects than these common primitive types. Perhaps you have a business object called `Employee`, which holds properties such as `Title`, `Name`, `Department`, and `Office`. Instead of configuring all these properties, you can instead just specify the `Employee` object as a property.

Then with code in either your code-behind classes or classes held with the `App_Code` folder, you can access this property as `Profile.Employee`. Thanks to the dynamic compiler, which is part of the ASP.NET runtime, these properties are immediately available within Visual Studio with IntelliSense support. That sounds really powerful, doesn't it? It is.

However, you can quickly paint yourself into a corner if you set up many complex objects as Profile properties. What happens when you need to add or remove properties to the `Employee` object but you already have many serialized versions of the object held in the database? How will you upgrade them?

When ASP.NET 2.0 first came out, I saw all kinds of examples of how you could create an object called `ShoppingBasket`, add objects called `Items` to the `ShoppingBasket`, and set up the `ShoppingBasket` as a Profile property. I immediately knew that would not be something I was going to attempt. In the year previous to the launch of .NET 2.0, I was building a commerce website to manage a basket with items and customer orders, and I never used the Profile properties. I still had my `ShoppingBasket` object, which held lots of `Item` objects, but I managed the tables and stored procedures used with those objects so that I could add new properties to them and easily manage the changes. If the `Item` started out with just one value for price called `Price`, and later I needed to add a couple more such as `SalesPrice` and `SeasonalPrice`, it was easy enough to add them. And to associate them to the current website user, I used the following technique.

For this particular website, I had to work with anonymous and authenticated users. When a user first came to the site, that user was given a token to identify him as he moved from page to page. This was an anonymous user token. After the user created an account, that anonymous token would be deleted and the user would be migrated to an authenticated user token. Authenticated users were Members with the Membership provider. To seamlessly work with anonymous and authenticated users, I needed a way to uniquely identify these users. Unfortunately, there is no such default value as `Profile.UserID`. There is a property for `Profile.UserName`, but that is available only for authenticated users. So I ended up creating a wrapper property, which provided a `Guid` value whether the user was anonymous or authenticated.

ANONYMOUS PROFILES

To make use of properties on the profile, you must first have anonymous profiles enabled. An element in `system.web` named `anonymousIdentification` can be enabled to turn on this feature.

In the `App_Code` folder, I created a class called `Utility` and added the properties in Listing 1-10.

Listing 1-10. *Utility Methods*

```
public static bool IsUserAuthenticated
{
  get
  {
    return HttpContext.Current.User.Identity.IsAuthenticated;
  }
}

public static Guid UserID
{
  get
  {
    if (IsUserAuthenticated)
    {
      return (Guid)Membership.GetUser().ProviderUserKey;
    }
    else
    {
      Guid userId =
        new Guid(HttpContext.Current.Request.AnonymousID.Substring(0, 36));
      return userId;
    }
  }
}fcdd
```

Whenever I needed the unique identifier for the current user, I would access it with `Utility.UserID`. But then I had to handle the transition from anonymous to authenticated. To do so, I added a method to `Global.asax` called `Profile_OnMigrateAnonymous`, shown in Listing 1-11.

Listing 1-11. *Profile_OnMigrateAnonymous*

```
public void Profile_OnMigrateAnonymous(object sender, ProfileMigrateEventArgs args)
{
  Guid anonID = new Guid(args.AnonymousID);
  Guid authId = (Guid)Membership.GetUser().ProviderUserKey;
```

```
  // migrate anonymous user resources to the authenticated user

  // remove the anonymous user.
  Membership.DeleteUser(args.AnonymousID, true);
}
```

In the case of the basket, I just added the items that were in the anonymous user's basket into the authenticated user's basket, and removed the anonymous account and all associated data.

I have considered adding a `Guid` property to the Profile properties called `UserID`, but the dynamic compiler for the ASP.NET runtime works only in code-behind files. It would not work in the `Utility` class held in the `App_Code` directory, which is where I place a good deal of the code. So the preceding technique was the only option.

Creating Users and Roles

When you work on a website with Visual Studio, you are able to use the Website Administration tool to create users and roles. But this utility is not a feature built into Microsoft Internet Information Server (IIS). To work with users and roles, you have to do it yourself—either in the database by carefully calling stored procedures or by creating an interface to safely use the Membership API. I chose to create a couple of user controls that I can easily drop into any website.

The two main controls, `UserManager.ascx` and `RolesManager.ascx`, do the work to manage the users and roles. These controls are held in another user control called `MembersControl.ascx`. This control switches between three views to create a new user, edit existing users, and edit roles. Listings 1-12 through 1-17 provide the full source for these controls.

Listing 1-12. *UserManager.ascx*

```
<%@ Control Language="C#" AutoEventWireup="true"
  CodeFile="UserManager.ascx.cs" Inherits="MemberControls_UserManager" %>
<asp:Label ID="TitleLabel"
  runat="server" Text="User Manager"></asp:Label><br />
<asp:MultiView ID="UsersMultiView" runat="server"
  ActiveViewIndex="0">
    <asp:View ID="SelectUserView" runat="server"
      OnActivate="SelectUserView_Activate">

        <table>
        <tr>
        <td>
            <asp:GridView ID="UsersGridView" runat="server"
                AllowPaging="True"
                AutoGenerateColumns="False"
                OnInit="UsersGridView_Init"
                OnPageIndexChanging="UsersGridView_PageIndexChanging"
                OnRowCommand="UsersGridView_RowCommand"
```

```
            GridLines="None">
            <Columns>
                <asp:BoundField DataField="UserName" HeaderText="Username" />
                <asp:BoundField DataField="Email" HeaderText="Email" />
                <asp:TemplateField ShowHeader="False">
                    <ItemTemplate>
                        <asp:LinkButton ID="LinkButton1" runat="server"
                            CausesValidation="false"
                            CommandName="ViewUser"
                            CommandArgument='<%# Bind("UserName") %>'
                            Text="View"></asp:LinkButton>
                    </ItemTemplate>
                </asp:TemplateField>
            </Columns>
            <RowStyle CssClass="EvenRow" />
            <HeaderStyle CssClass="HeaderRow" />
            <AlternatingRowStyle CssClass="OddRow" />
        </asp:GridView>
    </td>
    </tr>
    <tr>
    <td align="center">
        <asp:TextBox ID="FilterUsersTextBox" runat="server"
            Width="75px"></asp:TextBox>
        <asp:Button ID="FilterUsersButton" runat="server"
            OnClick="FilterUsersButton_Click" Text="Filter" />
    </td>
    </tr>
    </table>

</asp:View>
<asp:View ID="UserView" runat="server" OnActivate="UserView_Activate">

<table>
    <tr>
        <td class="Label">
            <asp:Label ID="UserNameLabel" runat="server" Text="User Name: "
                Font-Bold="True"></asp:Label></td>
        <td class="Data">
            <asp:Label ID="UserNameValueLabel" runat="server"
                Text=""></asp:Label></td>
    </tr>
    <tr>
        <td class="Label">
            <asp:Label ID="ApprovedLabel" runat="server" Text="Approved: "
                Font-Bold="True"></asp:Label></td>
        <td class="Data">
```

```
            <asp:Label ID="ApprovedValueLabel" runat="server"
                Text=""></asp:Label></td>
        </tr>
        <tr>
            <td class="Label">
                <asp:Label ID="LockedOutLabel" runat="server" Text="Locked Out: "
                    Font-Bold="True"></asp:Label></td>
            <td class="Data">
                <asp:Label ID="LockedOutValueLabel" runat="server"
                    Text=""></asp:Label></td>
        </tr>
        <tr>
            <td class="Label">
                <asp:Label ID="OnlineLabel" runat="server" Text="Online: "
                    Font-Bold="True"></asp:Label></td>
            <td class="Data">
                <asp:Label ID="OnlineValueLabel" runat="server"
                    Text=""></asp:Label></td>
        </tr>
        <tr>
            <td class="Label">
                <asp:Label ID="CreationLabel" runat="server" Text="Creation: "
                    Font-Bold="True"></asp:Label></td>
            <td class="Data">
                <asp:Label ID="CreationValueLabel" runat="server"
                    Text=""></asp:Label></td>
        </tr>
        <tr>
            <td class="Label">
                <asp:Label ID="LastActivityLabel" runat="server"
                    Text="Last Activity:" Font-Bold="True"></asp:Label></td>
            <td class="Data">
                <asp:Label ID="LastActivityValueLabel" runat="server"
                    Text=""></asp:Label></td>
        </tr>
        <tr>
            <td class="Label">
                <asp:Label ID="LastLoginLabel" runat="server"
                    Text="Last Login:" Font-Bold="True"></asp:Label></td>
            <td class="Data">
                <asp:Label ID="LastLoginValueLabel" runat="server"
                    Text=""></asp:Label></td>
        </tr>
        <tr>
            <td colspan="2" class="Data">
                <asp:Label ID="UserCommentLabel" runat="server"
                    Text="Comment:" Font-Bold="True"></asp:Label><br />
```

```
            <asp:Label ID="UserCommentValueLabel" runat="server"
                Text=""></asp:Label>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            <asp:Button ID="EditUserButton" runat="server"
                Text="Edit User" OnClick="EditUserButton_Click" />
            <asp:Button ID="ResetPasswordButton" runat="server"
                Text="Reset Password" OnClick="ResetPasswordButton_Click"
                Visible="False" />
            <asp:Button ID="UnlockUserButton" runat="server"
                OnClick="UnlockUserButton_Click" Text="Unlock" />
            <asp:Button ID="ReturnViewUserButton" runat="server"
                Text="Return" OnClick="ReturnViewUserButton_Click" />
        </td>
    </tr>
</table>

</asp:View>
<asp:View ID="EditorView" runat="server" OnActivate="EditorView_Activate">

<table>
    <tr>
        <td class="Label">
            <asp:Label ID="UserName2Label" runat="server" Text="User Name: "
              Font-Bold="True"></asp:Label></td>            <td class="Data">
            <asp:Label ID="UserNameValue2Label" runat="server" Text="">
            </asp:Label></td>
        <td>
             </td>
    </tr>
    <tr>
        <td class="Label">
            <asp:Label ID="EmailLabel" runat="server" Text="Email: "
              Font-Bold="True"></asp:Label>
        </td>
        <td class="Data">
            <asp:TextBox ID="EmailTextBox" runat="server"
              AutoCompleteType="Email"></asp:TextBox></td>
        <td>
            <asp:RequiredFieldValidator
              ID="EmailRequiredFieldValidator" runat="server"
              ErrorMessage="*"
              ControlToValidate="EmailTextBox"
              EnableClientScript="False"></asp:RequiredFieldValidator>
            <asp:RegularExpressionValidator
```

```
                    ID="RegularExpressionValidator1" runat="server"
                    ControlToValidate="EmailTextBox"
                      EnableClientScript="False"
                      ErrorMessage="*"
                      ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+) ➥
*\.\w+([-.]\w+)*"></asp:RegularExpressionValidator>
                </td>
        </tr>
        <tr>
            <td class="Label">
                <asp:Label ID="CommentLabel" runat="server" Text="Comment: "
                  Font-Bold="True"></asp:Label>
            </td>
            <td class="Data">
                <asp:TextBox ID="CommentTextBox" runat="server"
                  AutoCompleteType="Email" TextMode="MultiLine"></asp:TextBox></td>
            <td>
                 </td>
        </tr>
        <tr>
        <td class="Label"><asp:Label ID="Approved2Label" runat="server"
          Text="Approved: " Font-Bold="True"></asp:Label></td>
        <td class="Data"><asp:CheckBox ID="ApprovedCheckBox" runat="server" /></td>
        <td></td>
        </tr>
        <tr>
        <td class="Label"><asp:Label ID="RolesLabel" runat="server"
          Text="Roles " Font-Bold="True"></asp:Label></td>
            <td class="Data">
                <asp:CheckBoxList ID="RolesCheckBoxList" runat="server">
                </asp:CheckBoxList>
            </td>
        <td></td>
        </tr>
        <tr>
            <td colspan="3">
                <asp:Button ID="UpdateUserButton" runat="server"
                    Text="Update User" OnClick="UpdateUserButton_Click" />
                <asp:Button ID="CancelEditUserButton" runat="server"
                    OnClick="CancelEditUserButton_Click"
                    Text="Cancel" /></td>
        </tr>
    </table>

    </asp:View>
</asp:MultiView>
```

Listing 1-13. *UserManager.ascx.cs*

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class MemberControls_UserManager : UserControl
{
    #region "  Events  "
    protected void Page_Init(object sender, EventArgs e)
    {
    }
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void FilterUsersButton_Click(object sender, EventArgs e)
    {
        BindUsersGridView();
    }
    protected void EditUserButton_Click(object sender, EventArgs e)
    {
        UsersMultiView.SetActiveView(EditorView);
    }
    protected void UnlockUserButton_Click(object sender, EventArgs e)
    {
        MembershipUser user = CurrentUser;
        if (user != null)
        {
            user.UnlockUser();
            Membership.UpdateUser(user);
            BindUserView();
        }
    }
    protected void ResetPasswordButton_Click(object sender, EventArgs e)
    {
        MembershipUser user = CurrentUser;
        if (user != null)
        {
            user.ResetPassword();
        }
        UsersMultiView.SetActiveView(UserView);
    }
    protected void ReturnViewUserButton_Click(object sender, EventArgs e)
    {
```

```
    UsersMultiView.SetActiveView(SelectUserView);
}
protected void UpdateUserButton_Click(object sender, EventArgs e)
{
    if (CurrentUser != null)
    {
        MembershipUser user = CurrentUser;
        user.Email = EmailTextBox.Text;
        user.Comment = CommentTextBox.Text;
        user.IsApproved = ApprovedCheckBox.Checked;
        Membership.UpdateUser(user);

        foreach (ListItem listItem in RolesCheckBoxList.Items)
        {
            string role = listItem.Value;
            if (Roles.RoleExists(role))
            {
                if (!listItem.Selected &&
                    Roles.IsUserInRole(user.UserName, role))
                {
                    Roles.RemoveUserFromRole(user.UserName, role);
                }
                else if (listItem.Selected &&
                    !Roles.IsUserInRole(user.UserName, role))
                {
                    Roles.AddUserToRole(user.UserName, role);
                }
            }
        }

        UsersMultiView.SetActiveView(UserView);
    }
}
protected void CancelEditUserButton_Click(object sender, EventArgs e)
{
    UsersMultiView.SetActiveView(UserView);
}
protected void CancelRolesButton_Click(object sender, EventArgs e)
{
    UsersMultiView.SetActiveView(UserView);
}
protected void SelectUserView_Activate(object sender, EventArgs e)
{
    BindUsersGridView();
}
protected void UserView_Activate(object sender, EventArgs e)
{
```

```
    BindUserView();
}
protected void EditorView_Activate(object sender, EventArgs e)
{
    BindEditorView();
}
protected void UsersGridView_Init(object sender, EventArgs e)
{
}

protected void UsersGridView_PageIndexChanging(
  object sender, GridViewPageEventArgs e)    {
    UsersGridView.PageIndex = e.NewPageIndex;
    BindUsersGridView();
}
protected void UsersGridView_RowCommand(
  object sender, GridViewCommandEventArgs e)
{
    if ("ViewUser".Equals(e.CommandName))
    {
        CurrentUser = GetUser(e.CommandArgument.ToString());
        UsersMultiView.SetActiveView(UserView);
    }
}
#endregion

#region "  Methods  "
private void BindUsersGridView()
{
    if (String.Empty.Equals(FilterUsersTextBox.Text.Trim()))
    {
        UsersGridView.DataSource = Membership.GetAllUsers();
    }
    else
    {
        List<MembershipUser> filteredUsers = new List<MembershipUser>();
        string filterText = FilterUsersTextBox.Text.Trim();
        foreach (MembershipUser user in Membership.GetAllUsers())
        {
            if (user.UserName.Contains(filterText) ||
                user.Email.Contains(filterText))
            {
                filteredUsers.Add(user);
            }
        }
        UsersGridView.DataSource = filteredUsers;
    }
```

```
        UsersGridView.DataBind();
    }
    private void BindUserView()
    {
        MembershipUser user = CurrentUser;
        if (user != null)
        {
            UserNameValueLabel.Text = user.UserName;
            ApprovedValueLabel.Text = user.IsApproved.ToString();
            LockedOutValueLabel.Text = user.IsLockedOut.ToString();
            OnlineValueLabel.Text = user.IsOnline.ToString();
            CreationValueLabel.Text = user.CreationDate.ToString("d");
            LastActivityValueLabel.Text = user.LastActivityDate.ToString("d");
            LastLoginValueLabel.Text = user.LastLoginDate.ToString("d");
            UserCommentValueLabel.Text = user.Comment;

            UnlockUserButton.Visible = user.IsLockedOut;
            ResetPasswordButton.Attributes.Add("onclick",
              "return confirm('Are you sure?');");
        }
    }
    private void BindEditorView()
    {
        MembershipUser user = CurrentUser;
        if (user != null)
        {
            UserNameValue2Label.Text = user.UserName;
            EmailTextBox.Text = user.Email;
            CommentTextBox.Text = user.Comment;
            ApprovedCheckBox.Checked = user.IsApproved;

            RolesCheckBoxList.Items.Clear();
            foreach (string role in Roles.GetAllRoles())
            {
                ListItem listItem = new ListItem(role);
                listItem.Selected = Roles.IsUserInRole(user.UserName, role);
                RolesCheckBoxList.Items.Add(listItem);
            }
        }
    }

    public void Reset()
    {
        UsersMultiView.SetActiveView(SelectUserView);
        Refresh();
    }
```

```
public void Refresh()
{
    BindUsersGridView();
}

public bool IsUserAuthenticated
{
    get
    {
        return HttpContext.Current.User.Identity.IsAuthenticated;
    }
}
public string GetUserName()
{
    if (IsUserAuthenticated)
    {
        return HttpContext.Current.User.Identity.Name;
    }
    return String.Empty;
}
public MembershipUser GetUser(string username)
{
    return Membership.GetUser(username);
}

#endregion

#region "  Properties  "
[Category("Appearance"), Browsable(true), DefaultValue("User Manager")]
public string Title
{
    get
    {
        return TitleLabel.Text;
    }
    set
    {
        TitleLabel.Text = value;
        EnsureChildControls();
    }
}
[Category("Appearance"), Browsable(true), DefaultValue(false)]
public bool TitleBold
{
    get
    {
        return TitleLabel.Font.Bold;
```

```
        }
        set
        {
            TitleLabel.Font.Bold = value;
            EnsureChildControls();
        }
    }

    [Browsable(false)]
    public CssStyleCollection TitleStyle
    {
        get
        {
            return TitleLabel.Style;
        }
    }
    private MembershipUser CurrentUser
    {
        get
        {
            return ViewState["CurrentUser"] as MembershipUser;
        }
        set
        {
            ViewState["CurrentUser"] = value;
        }
    }
    #endregion
}
```

Listing 1-14. *RoleManager.ascx*

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="RolesManager.ascx.cs"
    Inherits="MemberControls_RolesManager" %>
<asp:Label ID="TitleLabel" runat="server"
  Text="Roles Manager" Font-Bold="true"></asp:Label>
<table>
    <tr>
        <td align="center">
            <asp:GridView ID="RolesGridView" runat="server"
                AutoGenerateColumns="False"
                OnRowCommand="RolesGridView_RowCommand"
                OnRowDataBound="RolesGridView_RowDataBound"
                GridLines="None"
                Width="100%">
                <Columns>
                    <asp:TemplateField HeaderText="Role">
                        <ItemTemplate>
```

```
                        <asp:Label ID="Label1" runat="server"
                            Text="Role"></asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:TemplateField HeaderText="Users">
                    <ItemTemplate>
                        <asp:Label ID="Label1" runat="server"
                            Text="Users"></asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:ButtonField CommandName="RemoveRole"
                    Text="Remove" />
            </Columns>
            <EmptyDataTemplate>
                 - No Roles -
            </EmptyDataTemplate>
            <RowStyle CssClass="EvenRow" />
            <AlternatingRowStyle CssClass="OddRow" />
            <HeaderStyle CssClass="HeaderRow" />
        </asp:GridView>
    </td>
</tr>
<tr>
    <td align="center">
        <asp:Label ID="Label3" runat="server"
            Font-Bold="True" Text="Role: "></asp:Label>
        <asp:TextBox ID="AddRoleTextBox" runat="server"
            Width="75px"></asp:TextBox>
        <asp:Button ID="AddRoleButton" runat="server"
            Text="Add" OnClick="AddRoleButton_Click" /></td>
</tr>
</table>
```

Listing 1-15. *RoleManager.ascx.cs*

```
using System;
using System.ComponentModel;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class MemberControls_RolesManager : UserControl
{
    #region "  Events  "

    protected void Page_PreRender(object sender, EventArgs e)
    {
        BindRolesGridView();
```

```
    }
    protected void RolesGridView_RowDataBound(object sender, GridViewRowEventArgs e)
    {
        if (e.Row.RowType == DataControlRowType.DataRow)
        {
            string role = e.Row.DataItem as string;
            foreach (TableCell cell in e.Row.Cells)
            {
                foreach (Control control in cell.Controls)
                {
                    Label label = control as Label;
                    if (label != null)
                    {
                        if ("Role".Equals(label.Text))
                        {
                            label.Text = role;
                        }
                        else if ("Users".Equals(label.Text))
                        {
                            label.Text = Roles.GetUsersInRole(role). ➥
Length.ToString();
                        }
                    }
                    else
                    {
                        LinkButton button = control as LinkButton;
                        if (button != null)
                        {
                            button.Enabled = Roles.GetUsersInRole(role).Length == 0;
                            if (button.Enabled)
                            {
                                button.CommandArgument = role;
                                button.Attributes.Add("onclick",
                                  "return confirm('Are you sure?');");
                            }
                        }
                    }
                }
            }
        }
    }
    protected void RolesGridView_RowCommand(
      object sender, GridViewCommandEventArgs e)
    {
        if ("RemoveRole".Equals(e.CommandName))
        {
            string role = e.CommandArgument as string;
            Roles.DeleteRole(role, true);
```

```
            BindRolesGridView();
        }
    }
    protected void AddRoleButton_Click(object sender, EventArgs e)
    {
        if (Page.IsValid)
        {
            string role = AddRoleTextBox.Text;
            if (!Roles.RoleExists(role))
            {
                Roles.CreateRole(role);
                AddRoleTextBox.Text = String.Empty;
                BindRolesGridView();
            }
        }
    }

    #endregion
    #region "  Methods  "

    public void Refresh()
    {
        BindRolesGridView();
    }
    private void BindRolesGridView()
    {
        RolesGridView.DataSource = Roles.GetAllRoles();
        RolesGridView.DataBind();
    }

    #endregion
    #region "  Properties  "

    [Category("Appearance"), Browsable(true), DefaultValue("Roles Manager")]
    public string Title
    {
        get
        {
            return TitleLabel.Text;
        }
        set
        {
            TitleLabel.Text = value;
        }
    }

    #endregion
}
```

Listing 1-16. *MembersControl.ascx*

```
<%@ Control Language="C#" AutoEventWireup="true"
  CodeFile="MembersControl.ascx.cs"
  Inherits="MembersControl" %>
<%@ Register Src="UserManager.ascx" TagName="UserManager" TagPrefix="uc2" %>
<%@ Register Src="RolesManager.ascx" TagName="RolesManager" TagPrefix="uc1" %>

    <br />
    <br />
    <b>Select View:</b>
    <asp:DropDownList ID="NavDropDownList" runat="server"
        AutoPostBack="True"
        OnSelectedIndexChanged="NavDropDownList_SelectedIndexChanged">
        <asp:ListItem>Create User</asp:ListItem>
        <asp:ListItem>Manage Users</asp:ListItem>
        <asp:ListItem>Manage Roles</asp:ListItem>
    </asp:DropDownList><br />
    <br />

    <asp:MultiView ID="MultiView1" runat="server">
    <asp:View ID="UserCreationView" runat="server"
      OnActivate="UserCreationView_Activate">
                <asp:CreateUserWizard ID="CreateUserWizard1" runat="server"
                    AutoGeneratePassword="True"
                    LoginCreatedUser="False"
                    OnCreatedUser="CreateUserWizard1_CreatedUser">
                    <WizardSteps>
                        <asp:CreateUserWizardStep
                          ID="CreateUserWizardStep1" runat="server">
                        </asp:CreateUserWizardStep>
                        <asp:CompleteWizardStep
                          ID="CompleteWizardStep1" runat="server">
                        </asp:CompleteWizardStep>
                    </WizardSteps>
                </asp:CreateUserWizard>
    </asp:View>
    <asp:View ID="UserManagerView" runat="server"
      OnActivate="UserManagerView_Activate">

                <uc2:UserManager
                  ID="UserManager1" runat="server"
                  Title="Users"
                  TitleBold="true" />
    </asp:View>
    <asp:View ID="RolesManagerView" runat="server">

                <uc1:RolesManager
```

```
                ID="RolesManager1" runat="server"
                Title="Roles" />

    </asp:View>
    </asp:MultiView>
```

Listing 1-17. *MembersControl.ascx.cs*

```
using System;
using System.Web.UI;

public partial class MembersControl : UserControl
{

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            MultiView1.SetActiveView(UserManagerView);
            NavDropDownList.SelectedValue = "Manage Users";
        }
    }

    protected void NavDropDownList_SelectedIndexChanged(object sender, EventArgs e)
    {
        if ("Create User".Equals(NavDropDownList.SelectedValue))
        {
            MultiView1.SetActiveView(UserCreationView);
        }
        else if ("Manage Users".Equals(NavDropDownList.SelectedValue))
        {
            MultiView1.SetActiveView(UserManagerView);
            RefreshUserManager();
        }
        else if ("Manage Roles".Equals(NavDropDownList.SelectedValue))
        {
            MultiView1.SetActiveView(RolesManagerView);
            RefreshRolesManager();
        }
    }

    protected void CreateUserWizard1_CreatedUser(object sender, EventArgs e)
    {
        MultiView1.SetActiveView(UserManagerView);
        NavDropDownList.SelectedValue = "Manage Users";
        RefreshUserManager();
    }
```

```
    private void RefreshUserManager()
    {
        MemberControls_UserManager userManager =
            UserManagerView.FindControl("UserManager1")
                as MemberControls_UserManager;
        if (userManager != null)
        {
            userManager.Refresh();
        }
    }

    private void RefreshRolesManager()
    {
        MemberControls_RolesManager rolesManager =
            UserManagerView.FindControl("RolesManager1")
              as MemberControls_RolesManager;
        if (rolesManager != null)
        {
            rolesManager.Refresh();
        }
    }

    protected void UserManagerView_Activate(object sender, EventArgs e)
    {
        UserManager1.Reset();
    }

    protected void UserCreationView_Activate(object sender, EventArgs e)
    {
        CreateUserWizard1.ActiveStepIndex = 0;
    }
}
```

Securing the Admin Section

With the user and role management controls placed into a folder named `Admin`, it can be secured by requiring authenticated users in the `Admin` role. Near the end of `Web.config`, you can create a location configuration for the `Admin` path and allow members in the `Admin` role (see Listing 1-18).

Listing 1-18. *Securing the Admin Section with Web.config*

```
<?xml version="1.0"?>

<configuration>

  <!-- other configuration settings -->
```

```
  <location path="Admin">
    <system.web>
      <authorization>
        <allow roles="Admin"/>
        <deny users="*"/>
      </authorization>
    </system.web>
  </location>

</configuration>
```

Creating the Admin User

After your `Admin` section is ready, you will naturally need the `Admin` user so you can log in to this section and use the controls. It is sort of a catch-22 scenario. But just as you can use the Membership API to manage users and roles, with these controls you can also programmatically create users and roles. To automatically ensure that your website has the necessary `Admin` user, you can add the code to do all of this work to the `Application_Start` event handler in the `Global.asax` file for the website. I first check whether the three default roles exist and add each one that does not exist. And then if there are no users, I have the method add the default `Admin` user, as shown in Listing 1-19.

Listing 1-19. *Adding Roles and Users*

```
public void Application_Start(object sender, EventArgs e)
{
  if (Roles.Enabled)
  {
    String[] requiredRoles = { "Admin", "Users", "Editors" };
    foreach (String role in requiredRoles)
    {
      if (!Roles.RoleExists(role))
      {
        Roles.CreateRole(role);
      }
    }
    string[] users = Roles.GetUsersInRole("Admin");
    if (users.Length == 0)
    {
      // create admin user
      MembershipCreateStatus status;
      Membership.CreateUser("admin", "CHANGE_ME", "admin@localhost",
            "Favorite color?", "green", true, out status);
      if (MembershipCreateStatus.Success.Equals(status))
      {
        Roles.AddUserToRole("admin", "Admin");
      }
      else
```

```
      {
        LogMessage("Unable to create admin user: " + status, true);
      }
    }
  }
}
```

Summary

This chapter covered how to prepare your environment for working with ASP.NET websites and how to configure the link to the database. You learned how to configure and manage the provider services, including adding users and roles programmatically so a new website can be managed immediately after it is deployed.