Implementing a Membership Provider

Other Versions

ASP.NET membership is designed to enable you to easily use a number of different membership providers for your ASP.NET applications. You can use the supplied membership providers that are included with the .NET Framework, or you can implement your own providers.

There are two primary reasons for creating a custom membership provider.

- You need to store membership information in a data source that is not supported by the membership providers included with the .NET Framework, such as a FoxPro database, an Oracle database, or other data sources.
- You need to manage membership information using a database schema that is different from the database schema used by the providers that ship with the .NET Framework. A common example of this would be membership data that already exists in a SQL Server database for a company or Web site.

Required Classes

To implement a membership provider, you create a class that inherits the MembershipProvider abstract class from the System.Web.Security namespace. The MembershipProvider abstract class inherits the ProviderBase abstract class from the System.Configuration.Provider namespace, so you must implement the required members of the ProviderBase class as well. The following tables list the required properties and methods that you must implement from the ProviderBase and MembershipProvider abstract classes and a description of each. To review an implementation of each member, see the code supplied for the Sample Membership Provider Implementation.

Required ProviderBase Members

Member	Description
Initialize method	Takes, as input, the name of the provider and a NameValueCollection of configuration settings. Used to set property values for the provider instance including implementation-specific values and options specified in the configuration file (Machine.config or Web.config) supplied in the configuration.

Required MembershipProvider Members

Member	Description

EnablePasswordReset property	A **Boolean** value specified in the configuration file (Web.config). The EnablePasswordReset property indicates whether users can use the ResetPassword method to overwrite their current password with a new, randomly generated password. This property is read-only.
EnablePasswordRetrieval property	A **Boolean** value specified in the configuration file (Web.config). The EnablePasswordRetrieval property indicates whether users can retrieve their password using the GetPassword method. This property is read-only.
RequiresQuestionAndAnswer property	A **Boolean** value specified in the configuration file (Web.config). The RequiresQuestionAndAnswer property indicates whether users must supply a password answer in order to retrieve their password using the GetPassword method, or reset their password using the ResetPassword method. This property is read-only.
RequiresUniqueEmail property	A **Boolean** value specified in the configuration file (Web.config). The RequiresUniqueEmail property indicates whether users must supply a unique e-mail address value when creating a user. If a user already exists in the data source for the current ApplicationName, the CreateUser method returns **null** and a status value of DuplicateEmail. This property is read-only.
PasswordFormat property	A MembershipPasswordFormat value specified in the configuration file (Web.config). The PasswordFormat property indicates the format that passwords are stored in. Passwords can be stored in **Clear**, **Encrypted**, and **Hashed** password formats. **Clear** passwords are stored in plain text, which improves the performance of password storage and retrieval but is less secure, as passwords are easily read if your data source is compromised. **Encrypted** passwords are encrypted when stored and can be decrypted for password comparison or password retrieval. This requires additional processing for password storage and retrieval but is more secure, as passwords are not easily determined if the data source is compromised. **Hashed** passwords are hashed using a one-way hash algorithm and a randomly generated salt value when stored in the database. When a password

	is validated, it is hashed with the salt value in the database for verification. Hashed passwords cannot be retrieved. You can use the EncryptPassword and DecryptPassword virtual methods of the MembershipProvider class to encrypt and decrypt password values, or you can supply your own encryption code. If you use the EncryptPassword and DecryptPassword virtual methods of the MembershipProvider class, **Encrypted** passwords are encrypted using the key information supplied in the machineKey element the configuration file. This property is read-only.
MaxInvalidPasswordAttempts property	An **Integer** value specified in the configuration file (Web.config). The MaxInvalidPasswordAttempts works in conjunction with the PasswordAttemptWindow to guard against an unwanted source guessing the password or password answer of a membership user through repeated attempts. If the number of invalid passwords or password questions supplied for a membership user exceeds the MaxInvalidPasswordAttempts within the number of minutes identified by the PasswordAttemptWindow, then the membership user is locked out by setting the IsLockedOut property to **true** until the user is unlocked using the UnlockUser method. If a valid password or password answer is supplied before the MaxInvalidPasswordAttempts is reached, the counter that tracks the number of invalid attempts is reset to zero. If the RequiresQuestionAndAnswer property is set to **false**, invalid password answer attempts are not tracked. Invalid password and password answer attempts are tracked in the ValidateUser, ChangePassword, ChangePasswordQuestionAndAnswer, GetPassword, and ResetPassword methods. This property is read-only.
PasswordAttemptWindow property	An **Integer** value specified in the configuration file (Web.config). For a description, see the description of the MaxInvalidPasswordAttempts property. This property is read-only.
ApplicationName property	The name of the application using the membership information specified in the configuration file (Web.config). The ApplicationName is stored in the data source with related user information and used when querying for that information. See the section on the ApplicationName later in this topic for more information.

	This property is read/write and defaults to the ApplicationPath if not specified explicitly.
MembershipProvider.CreateUser method	Takes, as input, the name of a new user, a password, and an e-mail address and inserts a new user for the application into the data source. The CreateUser method returns a MembershipUser object that is populated with the information for the newly created user. The CreateUser method also defines an **out** parameter (in Visual Basic, you can use **ByRef**) that returns a MembershipCreateStatus value that indicates whether the user was successfully created, or a reason that the user was not successfully created. The CreateUser method raises the ValidatingPassword event if a MembershipValidatePasswordEventHandler has been specified, and continues or cancels the create-user action based on the results of the event. You can use the OnValidatingPassword virtual method to execute the specified MembershipValidatePasswordEventHandler.
UpdateUser method	Takes, as input, a MembershipUser object populated with user information and updates the data source with the supplied values.
DeleteUser method	Takes, as input, the name of a user and deletes that user's information from the data source. The DeleteUser method returns **true** if the user was successfully deleted; otherwise, **false**. An additional **Boolean** parameter is included to indicate whether related information for the user, such as role or profile information is also deleted.
ValidateUser method	Takes, as input, a user name and a password and verifies that the values match those in the data source. The ValidateUser method returns **true** for a successful user name and password match; otherwise, **false**.
GetUser method	Takes, as input, a unique user identifier and a **Boolean** value indicating whether to update the LastActivityDate value for the user to show that the user is currently online. The GetUser method returns a MembershipUser object populated with current values from the data source for the specified user. If the user name is not found in the data source, the GetUser method returns **null** (**Nothing** in Visual Basic).
GetUser method	Takes, as input, a user name and a **Boolean** value indicating whether to update the LastActivityDate value for the user to show that the user is currently online. The GetUser method returns a MembershipUser object populated with current values from the data source for the specified user. If the user name is not found in the data source, the GetUser method returns **null** (**Nothing** in Visual Basic).

GetAllUsers method	Returns a MembershipUserCollection populated with MembershipUser objects for all of the users in the data source. The results returned by GetAllUsers are constrained by the *pageIndex* and *pageSize* parameters. The *pageSize* parameter identifies the maximum number of MembershipUser objects to return in the MembershipUserCollection. The *pageIndex* parameter identifies which page of results to return, where 0 identifies the first page. The *totalRecords* parameter is an `out` parameter that is set to the total number of membership users. For example, if 13 users were in the database for the application, and the *pageIndex* value was 1 with a *pageSize* of 5, the MembershipUserCollection returned would contain the sixth through the tenth users returned. *totalRecords* would be set to 13.
GetNumberOfUsersOnline method	Returns an integer value that is the count of all the users in the data source where the LastActivityDate is greater than the current date and time minus the UserIsOnlineTimeWindow property. The UserIsOnlineTimeWindow property is an integer value specifying the number of minutes to use when determining whether a user is online.
ResetPassword method	Takes, as input, a user name and a password answer and generates a new, random password for the specified user. The ResetPassword method updates the user information in the data source with the new password value and returns the new password as a **string**. A convenient mechanism for generating a random password is the GeneratePassword method of the Membership class. The ResetPassword method ensures that the EnablePasswordReset property is set to **true** before performing any action. If the EnablePasswordReset property is **false**, a NotSupportedException is thrown. The ResetPassword method also checks the value of the RequiresQuestionAndAnswer property. If the RequiresQuestionAndAnswer property is **true**, the ResetPassword method checks the value of the supplied answer parameter against the stored password answer in the data source. If they do not match, a MembershipPasswordException is thrown. The ResetPassword method raises the ValidatingPassword event, if a MembershipValidatePasswordEventHandler has been specified, to validate the newly generated password and continues or cancels the reset-password action based on the results of the event. You can use the OnValidatingPassword virtual method to execute the specified MembershipValidatePasswordEventHandler.
GetPassword method	Takes, as input, a user name and a password answer and retrieves the password for that user from the data source and returns the password as a **string**.

	GetPassword ensures that the EnablePasswordRetrieval property is set to **true** before performing any action. If the EnablePasswordRetrieval property is **false**, an ProviderException is thrown. The GetPassword method also checks the value of the RequiresQuestionAndAnswer property. If the RequiresQuestionAndAnswer property is **true**, the GetPassword method checks the value of the supplied answer parameter against the stored password answer in the data source. If they do not match, a MembershipPasswordException is thrown.
GetUserNameByEmail method	Takes, as input, an e-mail address and returns the first user name from the data source where the e-mail address matches the supplied *email* parameter value. If no user name is found with a matching e-mail address, an empty string is returned. If multiple user names are found that match a particular e-mail address, only the first user name found is returned.
ChangePassword method	Takes, as input, a user name, a current password, and a new password, and updates the password in the data source if the supplied user name and current password are valid. The ChangePassword method returns **true** if the password was updated successfully; otherwise, **false**. The ChangePassword method raises the ValidatingPassword event, if a MembershipValidatePasswordEventHandler has been specified, and continues or cancels the change-password action based on the results of the event. You can use the OnValidatingPassword virtual method to execute the specified MembershipValidatePasswordEventHandler.
ChangePasswordQuestionAndAnswer method	Takes, as input, a user name, a password, a password question, and a password answer, and updates the password question and answer in the data source if the supplied user name and password are valid. The ChangePasswordQuestionAndAnswer method returns **true** if the password question and answer are updated successfully; otherwise, **false**. If the supplied user name and password are not valid, **false** is returned.
FindUsersByName method	Returns a list of membership users where the user name contains a match of the supplied *usernameToMatch* for the configured ApplicationName. For example, if the *usernameToMatch* parameter is set to "user," then the users "user1," "user2," "user3," and so on

	are returned. Wildcard support is included based on the data source. Users are returned in alphabetical order by user name. The results returned by FindUsersByName are constrained by the *pageIndex* and *pageSize* parameters. The *pageSize* parameter identifies the number of MembershipUser objects to return in the MembershipUserCollection. The *pageIndex* parameter identifies which page of results to return, where 1 identifies the first page. The *totalRecords* parameter is an `out` parameter that is set to the total number of membership users that matched the *usernameToMatch* value. For example, if 13 users were found where *usernameToMatch* matched part of or the entire user name, and the *pageIndex* value was 2 with a *pageSize* of 5, then the MembershipUserCollection would contain the sixth through the tenth users returned. *totalRecords* would be set to 13.
FindUsersByEmail method	Returns a list of membership users where the user name contains a match of the supplied *emailToMatch* for the configured ApplicationName. For example, if the *emailToMatch* parameter is set to "address@example.com," then users with the e-mail addresses "address1@example.com," "address2@example.com," and so on are returned. Wildcard support is included based on the data source. Users are returned in alphabetical order by user name. The results returned by FindUsersByEmail are constrained by the *pageIndex* and *pageSize* parameters. The *pageSize* parameter identifies the number of MembershipUser objects to return in the MembershipUserCollection collection. The *pageIndex* parameter identifies which page of results to return, where 1 identifies the first page. The *totalRecords* parameter is an `out` parameter that is set to the total number of membership users that matched the *emailToMatch* value. For example, if 13 users were found where *emailToMatch* matched part of or the entire user name, and the *pageIndex* value was 2 with a *pageSize* of 5, then the MembershipUserCollection would contain the sixth through the tenth users returned. *totalRecords* would be set to 13.
UnlockUser method	Takes, as input, a user name, and updates the field in the data source that stores the IsLockedOut property to **false**. The UnlockUser method returns **true** if the record for the membership user is updated successfully; otherwise **false**.

ApplicationName

Membership providers store user information uniquely for each application. This enables multiple ASP.NET applications to use the same data source without running into a conflict if duplicate user names are created. Alternatively, multiple ASP.NET applications can use the same user data source by specifying the same ApplicationName.

Because membership providers store user information uniquely for each application, you will need to ensure that your data schema includes the application name and that queries and updates also include the application name. For example, the following command is used to retrieve a user name from a database, based on the e-mail address, and ensures that the ApplicationName is included in the query.

```
SELECT Username FROM MyUserTable
  WHERE Email = 'someone@example.com' AND ApplicationName = 'MyApplication'
```

Custom Members

You may need to extend the membership provider interfaces with additional functionality not provided by the ProviderBase and MembershipProvider abstract classes. Any public members that you add to your membership provider will be accessible using the Provider or Providers property of the Membership class.

An example of this could be a `LockUser` method that sets the IsLockedOut property to **true**. The following example shows how to cast the Provider property, which exposes the default membership provider for an application, as a custom-provider type in order to call the custom `LockUser` method.

C# VB

```
MyCustomProvider p = (MyCustomProvider)Membership.Provider;
p.LockUser(username);
```

Thread Safety

For each membership provider specified in the configuration for an application, ASP.NET instantiates a single membership provider instance that is used for all of the requests served by an HttpApplication object. As a result, you can have multiple requests executing concurrently. ASP.NET does not ensure the thread safety of calls to your provider. You will need to write your provider code to be thread safe. For example, creating a connection to a database or opening a file for editing should be done within the member that is called, such as MembershipProvider.CreateUser, rather than opening a file or database connection when the Initialize method is called.

See Also

Reference
ValidatePasswordEventArgs
OnValidatingPassword
Concepts
Sample Membership Provider Implementation
Securing ASP.NET Site Navigation
Other Resources
Managing Users by Using Membership
ASP.NET Security

Dev centers
Windows
Office
Visual Studio
Microsoft Azure
More...
Learning resources
Microsoft Virtual Academy
Channel 9
Interoperability Bridges
MSDN Magazine
Community
Forums
Blogs
Codeplex
Support
Self support
Programs
BizSpark (for startups)
DreamSpark
Imagine Cup
United States (English)
Newsletter Privacy & cookies Terms of use Trademarks

<div></div>