Professional SharePoint 2007 Development

Chapter 8: Building Personalized Solutions

ISBN-10: 0-470-11756-7
ISBN-13: 978-0-470-11756-9

8 Building Personalized Solutions

By Brendon Schwartz

A *portal* is a single point of entry for a family of web sites. Often these sites have complex relationships, such as multiple tree structures or a complicated grid or network topology. The portal takes care of common tasks such as user login, authentication, authorization, and personalization. In addition, the portal typically provides common user interface artifacts such as master pages and themes, giving the underlying sites a common appearance that can be easily personalized to suit each user's tastes. Finally, most portals offer artifacts such as "breadcrumb" controls that simplify navigation into the underlying sites.

ASP.NET 2.0 contains most of the building blocks you need to construct a portal, and several starter kits are available to help do that, such as DotNetNuke. SharePoint 2007 (that is, WSS 3.0 and MOSS 2007) uses these ASP.NET features, so you can build portals that greatly simplify access to complex SharePoint site topologies.

This chapter introduces you to the key concepts of building a SharePoint portal site:

- ❑ Using built-in features such as profiles, colleagues, and audiences
- ❑ Creating custom applications by developing with the Profiles API
- ❑ Managing membership by designing your own custom people picker
- ❑ Working with audiences programmatically

Building Rich Portal Solutions on SharePoint

Portals were originally created by providers of large public Internet services, such as Yahoo and MSN, that manage thousands of content-rich sites under a common umbrella. However, as corporate intranets and extranets have become more complex, portals have become popular aspects of company infrastructure. They provide employees and business partners with easy access to information such as company news, contact lists, documents, and reports. In addition, portals serve as entry points to back-end business services such as purchasing, manufacturing, and accounting. Human resources departments are deploying portals that provide convenient access to public documents such as the employee handbook as well as private access to an employee's own records. Furthermore, many HR portals include self-help features for tasks such as updating information related to health insurance and retirement plans.

As the needs of these applications have grown, the applications running these sites have grown. They have become easier to customize and have started to use more powerful user data. The portals used by many companies take the form of many different applications today as follows:

- ❑ Standard sets of web pages
- ❑ Content Management Servers (CMS systems)
- ❑ User portal-based applications
- ❑ Internal portals for company use
- ❑ External portals for business-to-business use

Although it is difficult to be everything to everyone, Windows SharePoint Services provides a framework that Office SharePoint Server builds upon to build these types of portal systems. One of the most important features needed for a portal are the user profiles. Each user has a set of profile data that can be used to display information to other users or determine which content the user should see. The profile data can be managed easily from a central location and can be shared across multiple web applications. This data can be imported from other sources or used from a central source.

SharePoint also includes the My Site feature that enables each user to create a personal site with public and private views. This enables users to save, manage, and share information much more conveniently than older file-sharing techniques allowed. This chapter looks at how to manage My Site with the user interface as well as through the object model.

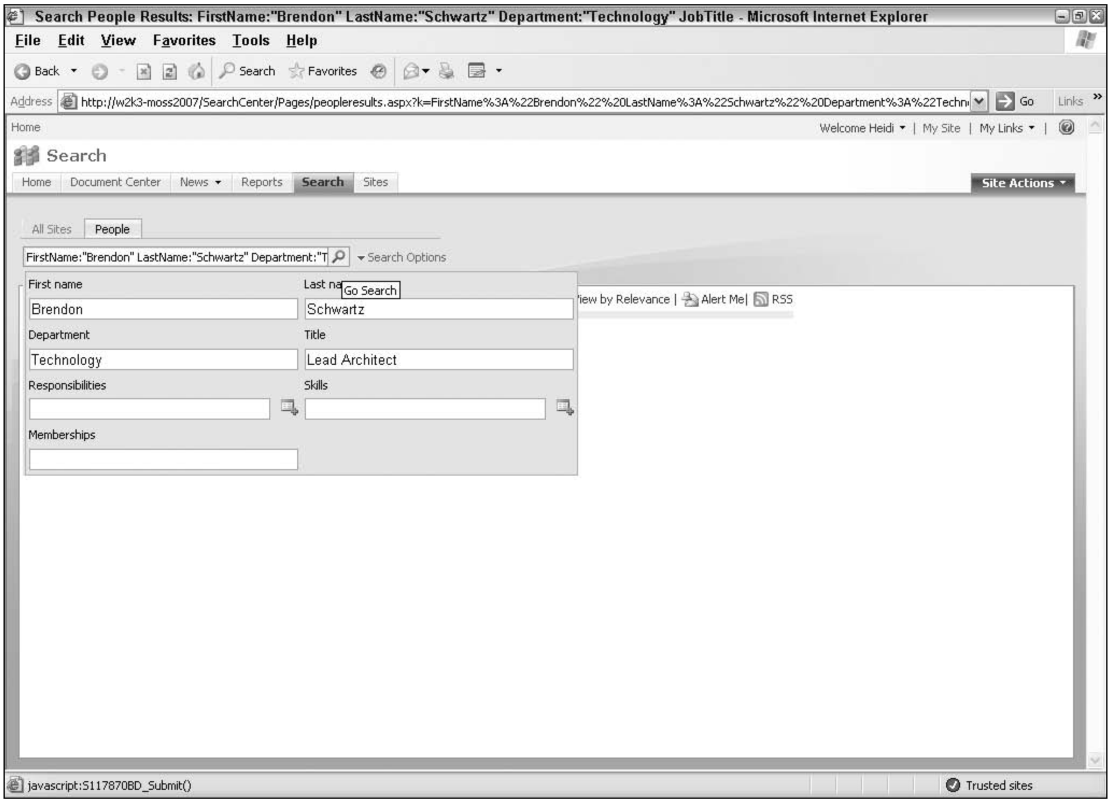
To make the sites dynamic, SharePoint uses audiences to determine the correct content to display to the right users. In addition to audiences, you can use specific Web Parts to show data about yourself and other Colleagues. This allows you to set up a group of users that match a certain profile to be shown the data they need immediately.

Membership and User Profiles

User profiles are a collection of information that can be imported from a database or manually entered. The data in a user profile can then also be further customized by the user. Profiles store personal information about the users in your company or organization. All of the user profiles are stored in Profile

Services, which is part of the Shared Service Provider for the site collection. User profiles can be helpful when you need to determine some type of personal information about another user. Also, user profiles help by pointing out the connections, or relationships, that a user has with other users. For example, a manager would be able to see other managers in the organization, and use them as a resource when help is needed.

You can use user profiles through the user interface or through the APIs provided by the object model. There are many ways that user profiles can be helpful to an organization. For example, if a department is trying to use an unfamiliar program, they could search all of the user profiles to find someone who lists using that product as one of their skills. If there are only a handful of people in the organization who have experience using that product, the user profiles would provide a way to find them quickly and get the help needed. Figure 8-1 shows an advanced search of user profiles.


Search People Results: FirstName:"Brendon" LastName:"Schwartz" Department:"Technology" JobTitle - Microsoft Internet Explorer

Address: http://w2k3-moss2007/SearchCenter/Pages/peopleresults.aspx?k=FirstName%3A%22Brendon%22%20LastName%3A%22Schwartz%22%20Department%3A%22Techn

FirstName:"Brendon" LastName:"Schwartz" Department:"T

First name	Last name
Brendon	Schwartz
Department	Title
Technology	Lead Architect
Responsibilities	Skills
Memberships	

javascript:S117870BD_Submit()

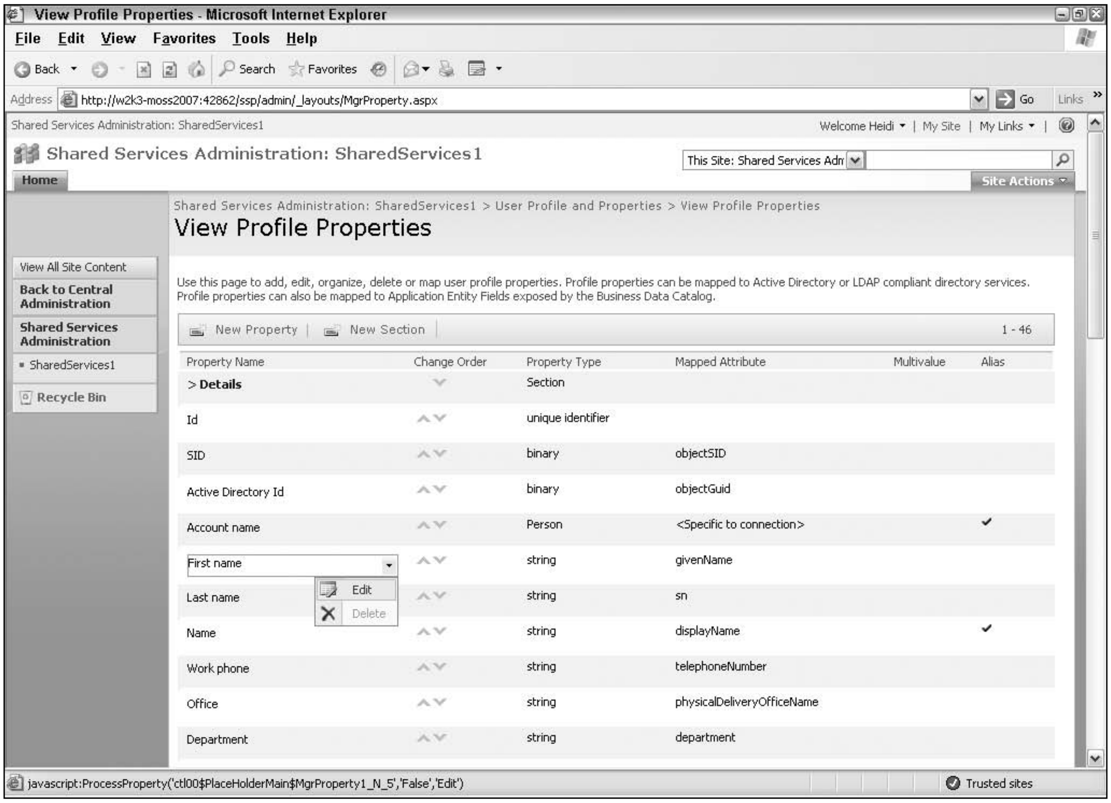
Figure 8-1

Managing a User Profile

To update the user profile, end users will use their My Site page, the administrator will use the Shared Service Provider, and you as the developer can access web services or the object model. When users set up a My Site page, there are five links provided in the Get Started with My Site section to help them personalize their site. The first link allows the user to describe themselves, and it takes the user to a personal profile form. The personal profile is where users can view and modify available user profile fields about themselves. Filling in the information is part of the process of creating memberships and helps other people find them through enterprise search, and also give colleagues some information about each other. This type of information can be very useful in large organizations, where many people don't know what others do or how to find them.

User profile data can be imported from many different connections such as Active Directory or another third-party data store. The administrator can manage the user profiles through the Shared Service Provider User Profiles and Properties screen. From this screen you can add, view, and create new imports for the user profile as well as manage the profile properties of each user. (See Figure 8-2.)

Although the administrator may already have some information set up to be displayed, each user should fill out all of the optional information to create a more robust site and user profile system. It is helpful for users to provide as much information as possible in their profiles, to increase the effectiveness of searches within your organization.


View Profile Properties - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://w2k3-moss2007:42862/ssp/admin/_layouts/MgrProperty.aspx
Go Links
Shared Services Administration: SharedServices1
Welcome Heidi | My Site | My Links
Shared Services Administration: SharedServices1
This Site: Shared Services Adm
Home
Site Actions
Shared Services Administration: SharedServices1 > User Profile and Properties > View Profile Properties
View All Site Content
Back to Central Administration
Shared Services Administration
SharedServices1
Recycle Bin
New Property | New Section
1 - 46
Edit
Delete
javascript:ProcessProperty('ctl00$PlaceHolderMain$MgrProperty1_N_5','False','Edit')
Trusted sites

View Profile Properties

Use this page to add, edit, organize, delete or map user profile properties. Profile properties can be mapped to Active Directory or LDAP compliant directory services. Profile properties can also be mapped to Application Entity Fields exposed by the Business Data Catalog.

Property Name	Change Order	Property Type	Mapped Attribute	Multivalue	Alias
> **Details**		Section			
Id		unique identifier			
SID		binary	objectSID		
Active Directory Id		binary	objectGuid		
Account name		Person	<Specific to connection>		✓
First name		string	givenName		
Last name		string	sn		
Name		string	displayName		✓
Work phone		string	telephoneNumber		
Office		string	physicalDeliveryOfficeName		
Department		string	department		

Figure 8-2

Setting Up Connections

To pull user profile information into SharePoint's user profile system, SharePoint sets up a connection to a data store that provides user information. This can include Active Directory, a Lightweight Directory Access Protocol (LDAP) server, or Business Data Catalog. There are two connection types used to bring the profile data into SharePoint.

- ❑ Master Connection
- ❑ Sub Connection

A master connection is used as the primary source of the profile data. The master connection is set up to use either Active Directory or LDAP. Sub connections are used to provide supplemental information to the user profile system of SharePoint. The Business Data Catalog can be used as a supplemental store.

User Profile API

The SharePoint Administration Site enables administrators to create and update user profiles as well as manage them from a central location. There are many times when more functionality is required, and custom code can be used to easily extend the user profiles. The primary namespace that was used in SharePoint 2003 for profiles was `Microsoft.SharePoint.Portal.UserProfiles`. This namespace has now been replaced by the `Microsoft.Office.Server.UserProfiles` namespace.

Each SharePoint web application uses a Shared Service Provider to manage the user profile data store. Because web applications can use the same Shared Service Provider, more than one SharePoint web application can use the same set of users stored in the User Profile section.

If you want to import a set of users from another application, SharePoint provides you with a few options. You can use the Business Data Catalog or the built-in APIs from the SharePoint object model, or even the user profile web services. If you are importing users on a regular basis and want to keep two systems synchronized, then the Business Data Catalog is probably the right solution. But if you are creating Web Parts or simple management tools, then the API or web services will provide you with the ability to create those.

To set up users in your SharePoint web application, you can either use the user interface that is provided or you can programmatically create users in custom code. You will look at how to create the users in code shortly. The two major classes that are used are:

- ❑ `UserProfile` class
- ❑ `UserProfileManager` class

Let's take a look at the classes that are used and how to work with the User Profile API in code.

Managing Users in the User Profile

`UserProfileManager` is the main object for accessing the user profile data stored on the web application's shared service. This class can be used to create users, delete users, and provide information about the profile service.

When working with the object model classes make sure to add a reference to the Windows SharePoint Services DLL as well as the required Microsoft Office SharePoint Server components' DLLs. If you don't have a reference to `System.Web.dll`, then you will need to add that also.

Creating a `UserProfileManager` is straightforward and can be done with the new syntax for SharePoint 2007. The following simple console application creates a `UserProfileManager` that is connected to the `ServerContext` of a SharePoint site. Then it displays the number of user profiles in the console window.

To create your own console application that displays the number of users in a site follow these steps:

1. Open Visual Studio, and create a new console application called `UserProfileManagerApp`.
2. Make sure to add the SharePoint references:

```
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Then in the main function, add the following code:

```
using System;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
namespace UserProfileManagerApp
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                using (SPSite spSite = new SPSite(@"http://localhost"))
                {
                    ServerContext siteContext = ServerContext.GetContext(spSite);
                    UserProfileManager pmManager =
                              new UserProfileManager(siteContext);
                    Console.WriteLine("Number of users: " + pmManager.Count);
                }
            }
            catch (Exception exp)
            {
                    Console.WriteLine(exp.Message);
                }
            }
        }
    }
```

In addition to the `Count` property, the `UserProfileManager` class contains methods to manage user profile data, including the following:

Name	Description
`CreateUserProfile`	Creates a user profile with the provided values. If the profile already exists, an exception is thrown.
`GetUserProfile`	Returns a reference to the user profile specified.
`RemoveUserProfile`	Removes user profiles. The method has multiple overrides to provide options when deleting users.
`UserExists`	Returns a value indicating whether the user for a specified account name has a profile.

Creating New User Profiles

Creating a new user in the user profile system only requires that the user name match an existing user in the current SharePoint site.

A good policy when creating user profiles is to check to make sure that the profile does not exist already. Although the profile system doesn't let you add a new profile if one already exists, it is much better practice to use the built-in methods for checking for the user, than to try to manage the exceptions that occur when trying to add the user a second time. In some cases, there are no built-in methods for performing a check and you must use the `try/catch` method. This is true when looking for a specific list in the object model.

The following code adds a new profile after checking to make sure that the user doesn't already have one:

1. Using the console application above, add the new code to check if a user exists before you create a new user and profile:

```
using System;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
namespace UserProfileManagerApp
{
    class Program
    {
            static void Main(string[] args)
            {
                try
                {
                    using (SPSite spSite = new SPSite(@"http://localhost"))
                    {
                        ServerContext siteContext = ServerContext.GetContext(spSite);
                        UserProfileManager pmManager =
                                  new UserProfileManager(siteContext);
                        Console.WriteLine("Number of users: " + pmManager.Count);
                        string strUserName = "wrox\\brendon";
                        UserProfile newUser = null;
                        if (!pmManager.UserExists(strUserName))
                        {
                            newUser = pmManager.CreateUserProfile(strUserName);
                        }
                    }
                }
                catch (Exception exp)
                {
                    Console.WriteLine(exp.Message);
                }
            }
        }
    }
```

Deleting User Profiles

The code for deleting profiles is very similar to the code for adding them, except that the method `RemoveUserProfile` is used. This method accepts either the user's name or the GUID that is the unique key for the user.

The following code removes the profile that was created in the preceding example. This process does not delete the user from the main connection such as Active Directory. Deleting the user from Active Directory will still have to be done from outside SharePoint.

Remember that you are also checking to see if the user profile exists before you can delete it. If the user profile does not exist, then you do not have to take any action.

```
    using System;
    using Microsoft.SharePoint;
    using Microsoft.Office.Server;
    using Microsoft.Office.Server.UserProfiles;
    namespace DeleteUser
    {
        class Program
        {
            static void Main(string[] args)
            {
                try
                {
                    using (SPSite spSite = new SPSite(@"http://localhost"))
                {
                    ServerContext siteContext = ServerContext.GetContext(spSite);
                    UserProfileManager pmManager =
                                   new UserProfileManager(siteContext);
                    Console.WriteLine("Number of users: " + pmManager.Count);
                    string strUserName = "wrox\\brendon";
                    if (pmManager.UserExists(strUserName))
                    {
                        pmManager.RemoveUserProfile(strUserName);
                    }
                }
            }
            catch (Exception exp)
            {
                Console.WriteLine(exp.Message);
            }
            Console.ReadLine();

        }
    }
}
```

Working with User Profiles

At the core of the user profile objects is the `UserProfile` class, which represents a single instance of a user profile. Once you have an object of this type, you can read and change the profile properties, assuming that the code is running with appropriate privileges.

To retrieve a profile, you just provide the user's name or GUID, as shown here:

1. Open Visual Studio and create a new console application called `RetrieveUser`.
2. Make sure to add the SharePoint references:

```
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Now get a reference to the correct SharePoint site and `UserProfileManager` associated with that site:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;

namespace RetrieveUser
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                using (SPSite spSite = new SPSite(@"http://localhost"))
                {
                    ServerContext siteContext = ServerContext.GetContext(spSite);
                    UserProfileManager pmManager =
                             new UserProfileManager(siteContext);
                }
            }
            catch (Exception exp)
            {
                Console.WriteLine(exp.Message);
            }
        }
    }
}
```

4. Check if the user exists, and get the user profile if the user is available:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;

namespace RetrieveUser
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                using (SPSite spSite = new SPSite(@"http://localhost"))
                {
                    ServerContext siteContext =
ServerContext.GetContext(spSite);
                    UserProfileManager pmManager =
                            new UserProfileManager(siteContext);
                    string strUserName = "devcow\\brendon";
                    UserProfile userProfile = null; [Author: Why is this variable
  called "newUser" when it actually holds the profile for an existing user? In fact,
  why isn't it called something like "userProfile" since that's what it is?i have
  updated the variable,thanks,brendon]
                    if (pmManager.UserExists(strUserName))
                    {
                        userProfile = pmManager.GetUserProfile(strUserName);
                    }
                }
            }
            catch (Exception exp)
            {
                    Console.WriteLine(exp.Message);
                }
            }
        }
    }
```

The `UserProfile` class contains many useful properties that can be leveraged when programming with the User Profile services of SharePoint 2007. The following table describes two properties used to identify the user profile and to return information about a given user profile.

Name	Description
ID	The unique GUID value of the current user profile. This value can be used when working with the `UserProfile` in the `UserProfileManager`.
Item	Contains a property value for the current user profile.

The `UserProfileManager` contains the list of user profile properties that are stored in the user profile data store. These properties are stored in the Shared Service Provider of the web site related to the `UserProfileManager`. To determine which properties are needed in the user profile store, use the `UserProfileManager` class. It is important to remember the difference between the name and the `DisplayName` when dealing with properties in SharePoint. The *name* is the value used when getting and setting the values, and the *DisplayName* is what is shown on the user interface. Sometimes these can be the same, but they are usually different. The `IsRequired` field shows which properties need to have values entered when the profile is updated in the user interface.

The following code sample demonstrates some of the default property values of the `UserProfileManager`. The code loops through all of the properties that have been created for the user profiles in the Shared Service. This will display the Name, Display Name, Description, and if the field is required in the console application you just created above.

```
    //Get the properties of the Property Manager
    foreach (Property property in pmManager.Properties)
    {
       Console.WriteLine("Name: " + property.Name);
       Console.WriteLine("DisplayName: " + property.DisplayName);
       Console.WriteLine("Description: " + property.Description);
       Console.WriteLine("Required: " + property.IsRequired);
    }
```

Setting User Profile Value Properties

User profile value properties can be entered and edited through the administrative screen, the user's My Site, or through code. The site administrator is also able to add properties to a user profile. These properties provide metadata that describes the user such as email, manager, and skills. You can have as many properties on a user to provide information about that user as needed. These properties can be set through an import from another system such as Active Directory or the Business Data Catalog.

Only the properties that have been created for the user have values, and the rest of the properties will not contain any value. It is good to check to make sure that the property exists for the user before trying to use the value.

The collection returned from the `UserProfile` class is a set of Name/Value pairs. Name is simply set as a string value that can be shown to the user, but Value is now a collection called `UserProfileValueCollection`. Value is a collection in SharePoint 2007 because the properties now have the ability to contain multivalues, which is something you will look at shortly. For now, just return the values as the value field.

The following code shows how you can loop through each value and display the results to the command window:

Using the `RetrieveUser` console application above, add the following lines to the main method:

```
foreach (KeyValuePair<string,UserProfileValueCollection> NameValues in userProfile)
{
   // Displays all of the name value pairs
   Console.WriteLine(NameValues.Key + ": " + NameValues.Value);
}
```

Now that you have seen what the values are for the properties of your user, you can add the correct values for the fields that are missing but required. You can also update the properties that are already set in the user profile.

Setting Single Value Properties

The standard type of property contained in the `UserProfile` class is a single value property. This property contains a Name/Value pair that has the information entered about the user.

To update the properties through the SharePoint object model follow these steps:

1. Create a console application called `UserProfileProperties`.
2. Add the references to SharePoint:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Connect to the user profile store and then retrieve a user from the data store. Once you have the user, you can use the indexer method to get the property that is expected. Add the following code to the main method to perform these actions:

```
            try
            {
                using (SPSite spSite = new SPSite(@"http://localhost"))
                {
                    ServerContext siteContext = ServerContext.GetContext(spSite);
                    UserProfileManager pmManager =
                            new UserProfileManager(siteContext);
                    string strUserName = "devcow\\brendon";
                    UserProfile userProfile = null;
                    if (pmManager.UserExists(strUserName))
                    {
                        userProfile = pmManager.GetUserProfile(strUserName);
                    }
                }
            }
            catch (Exception exp)
            {
                Console.WriteLine(exp.Message);
            }
```

4. Update a value such as Department. To persist the values back to the user profile system, you must call the `Commit()` method on the `UserProfile` that has new changes. This allows you to make many changes in memory and then commit all of the changes at once.

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;

namespace UserProfileProperties
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                using (SPSite spSite = new SPSite(@"http://localhost"))
                {
                    ServerContext siteContext = ServerContext.GetContext(spSite);
                    UserProfileManager pmManager =
                            new UserProfileManager(siteContext);
                    string strUserName = "devcow\\brendon";
                    UserProfile userProfile = null;
                    if (pmManager.UserExists(strUserName))
                    {
                        userProfile = pmManager.GetUserProfile(strUserName);
                        userProfile["Department"].Value = "Information Technology";
                        userProfile.Commit();
                    }
                }
            }
            catch (Exception exp)
            {
                    Console.WriteLine(exp.Message);
                }
            }
        }
    }
```

Setting the value requires knowing the string name of the field. SharePoint helps provide a base set of values used in the user profile system. These values are in a class called `PropertyConstants`. The `PropertyConstants` class only contains a set of constants that return the correct string value that is used for the name of the user profile property it refers to.

For example the value of `PropertyConstants.WorkPhone` is `WorkPhone`, whereas the value for `UserGuid` is `UserProfile_GUID`.

As you can see, some of the values are the same, but some can be different. The `PropertyConstants` class will help you retrieve and set all of the common properties in the user profile.

Look at how the following code uses the `PropertyConstants`, as well as a string, to set the value of each property.

5. Update another property, this time using the `PropertyConstants` class:

```
userProfile = pmManager.GetUserProfile(strUserName);
userProfile [PropertyConstants.WorkPhone].Value = "555-555-5555";
userProfile.Commit();
```

Multivalue Properties

A big improvement in SharePoint 2007 is the use of properties called multivalue properties. Sometimes you want the ability to store more than one value for a given field. This can come in very handy for multiple value properties such as sales area if you cover more than one area. Microsoft has some standard fields that are already set up for multivalue use such as Responsibilities, Skills, Past Projects, Interests, and Schools.

To determine whether the property is a multivalued property, the property `IsMultiValued` is provided. This value is a read-only property and can only be set by creating the user profile property as a multi-value property.

As you saw earlier, the multiple values have to be returned as a collection, and these values are returned as an `ArrayList` object. However, if you just display the properties value, the first element of the collection is the value that is displayed.

To add multiple values to the property, just call the `Add` method on the returned `ArrayList` property collection of the user:

```
Property skillProperty =
          pmManager.Properties.GetPropertyByName(PropertyConstants.Skills);
if (skillProperty.IsMultivalued)
{
   if (pmManager.UserExists(strUserName))
   {
      newUser = pmManager.GetUserProfile(strUserName);
```

```
            newUser[PropertyConstants.Skills].Add("ASP.NET");
            newUser[PropertyConstants.Skills].Add("SQL Server");
            newUser.Commit();
        }
    }
```

Each value in the user interface will be separated by a single character. The default character that is used is the comma. Some values might contain a comma, so SharePoint 2007 allows you to modify the separator character of a multivalue property. Use the `MultiValueSeparator` enumeration to select a different value for the property's `Separator` value. There are four values to choose from in the `MultiValueSeparator` class. The most common separator values are provided for you, as well as an `Unknown` value.

- Comma
- Semicolon
- Newline
- Unknown

When using the multivalue properties, you will need to pick which type of separator you will use. If the property type is a multivalue, set the separator:

```
    property.Separator = MultiValueSeparator.Semicolon;
```

Providing Properties in Choice Lists

In addition to having the ability to add multiple values to a property, you can also allow your users to select from a list of possible values known as a choice list. There are three different values that can be set for the ChoiceList property as follows:

- **None** — The property is not currently using a choice list.
- **Closed** — The choices have been added to the property and the users cannot edit the choices
- **Open** — Users and administrators can add new choices to the list.

User Profile Relationships

Some of the most common methods return information about the user or provide the application with more information about a user, such as relationships to other profiles. The `UserProfile` class is full of information about the user — such as lists of Managers and Peers — and provides the ability to create the user's personal site. The user will have access to this site by using the My Site link in the user interface of SharePoint.

The following table lists the `UserProfile` methods:

Name	Description
`Colleagues`	Returns a `ColleagueManager` to work with the current colleagues

Continued

Name	Description
`GetCommonManager`	Gets the user profile of the common managers of the current user
`GetDirectReports`	Returns a list of `UserProfiles` that contains the users that report to the current user
`GetManager`	Gets the current user's immediate manager
`GetManagers`	Returns a list of all the managers that the current user reports to
`GetPeers`	Gets a list of user profiles of all users that are the current user's peers
`Memberships`	Returns a list of the current user's memberships
`RefreshManagers`	Updates the list of managers the user has as associations

Adding the SharePoint People Picker to the Web Site

The people pickers in SharePoint make it easy to work with any of the users from the data source locations. To work with the control in the user interface, add the `PeopleEditor` control, as shown in the following code, to a Web Part or web page, and then you will have full control over how the people picker control is displayed and interacts with the user. You will be able to set properties of the `PeoplePicker` to modify how it is displayed. This is the same control that is used in a list or profile. The following code adds the `PeoplePicker` to a Web Part, but it could just as easily be added to a web page.

1. Start off by creating a new Web Part called `PeoplePickerWebPart`.
2. Add the required references:

```
using System;
using System.Runtime.InteropServices;
using System.Web.UI;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Serialization;

using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
using Microsoft.SharePoint.WebPartPages;
using System.Web.UI.WebControls;
```

3. In the `CreateChildControls` method create a new instance of the `PeopleEditor` control:

```
PeopleEditor PeoplePickerControl = new PeopleEditor();
this.Controls.Add(PeoplePickerControl);
base.CreateChildControls();
```

4. Update some of the properties of the control, such as allowing the control so that they are empty, allowing the end user to type in information, and changing the look and feel by changing the button names:

```
using System;
using System.Runtime.InteropServices;
using System.Web.UI;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Serialization;

using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
using Microsoft.SharePoint.WebPartPages;
using System.Web.UI.WebControls;

namespace PeoplePickerWebPart
{
    public class AgentPicker : System.Web.UI.WebControls.WebParts.WebPart
    {
        protected override void CreateChildControls()
        {
            try
            {
                PeopleEditor PeoplePickerControl = new PeopleEditor();
                PeoplePickerControl.AllowEmpty = true;
                PeoplePickerControl.AllowTypeIn = true;
                PeoplePickerControl.BrowseButtonImageName = "browse";
                PeoplePickerControl.BrowseButtonToolTip = "Pick an Agent";
                PeoplePickerControl.CheckButtonImageName = "Validate user";
                PeoplePickerControl.ErrorMessage = "No Agent Found";

                this.Controls.Add(PeoplePickerControl);
                base.CreateChildControls();
            }
            catch (Exception ex)
            {
                Literal _ErrorMessageLiteral = new Literal();
                _ErrorMessageLiteral.Text = "Custom Error: " + ex.Message;

                this.Controls.Clear();
                this.Controls.Add(_ErrorMessageLiteral);
            }
        }

    }
}
```

Developing Using the User Profile Change Log

When working with changes to used profiles, the `UserProfileChangeQuery` class provides the ability to filter based on the following types:

- `Add`
- `Anniversary`

- `Colleague`
- `Delete`
- `DistributionListMembership`
- `MultiValueProperty`
- `PersonalizationSite`
- `QuickLink`
- `SingleValueProperty`
- `SiteMembership`
- `Update`
- `UpdateMetadata`
- `UserProfile`
- `Weblog`

To filter, set one of these values to true once you have created an instance of the class. Multiple filter values can be set to true. The anniversary value will return all profile values that have dates in them. SharePoint considers each date field an anniversary and will create an anniversary event when the date reaches the machine date.

Each change type has a class that is associated with it that contains specific information for that type of change.

The `UserProfile` class contains methods to get changes about the user or the user's colleagues.

Name	Description
`CurrentChangeToken`	Represents a timestamp of the last time the current user was modified
`GetChanges`	Returns a collection of the changes that have happened to the current user
`GetColleagueChanges`	Returns a collection of changes that have happened to all of the colleagues associated with the current user

To get the change information, you must get the user profile and call the method `GetChanges` as follows:

1. Create a new console application named `ProfileChanges`.
2. Add the required SharePoint references:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Get a reference to the Site and User Profile Manager.

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

4. Determine and set the range for the number of days you want to look at in changes, then set the change token that will be used with that amount of time. This example uses the last 30 days.

```
// Display the changes that have occurred in the last month
DateTime dtNumberDays = DateTime.UtcNow.Subtract(TimeSpan.FromDays(30));
// Create a change token to compare the number of days
UserProfileChangeToken upChangeToken = new
        UserProfileChangeToken(dtNumberDays);
```

5. Now create a query object to determine which changes will be returned. For this example, set the `Anniversary` and `Colleague` changes to `true`, and set the `ChangeTokenStart` value from the change token created above.

```
// Create a query object to determine which changes are displayed
UserProfileChangeQuery QueryAnniversayAndColleagues = new
              UserProfileChangeQuery(false,true);
QueryAnniversayAndColleagues.ChangeTokenStart = upChangeToken;
QueryAnniversayAndColleagues.Anniversary = true;
QueryAnniversayAndColleagues.Colleague = true;
```

6. Now perform the standard steps to get a user profile, and this time call the method `GetChanges` with the query object you just created.

```
string strUserName = "heididev\\brendon";
if (pmManager.UserExists(strUserName))
{
    UserProfile spMyUser = pmManager.GetUserProfile(strUserName);

    UserProfileChangeCollection MyUserProfileChanges =
                spMyUser.GetChanges(QueryAnniversayAndColleagues);
}
```

7. Loop through each change in the returned collection of user profile changes, determine what type of change it is, and display information about that type of change:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
```

```
        {
           ServerContext siteContext = ServerContext.GetContext(spSite);
           UserProfileManager pmManager = new UserProfileManager(siteContext);

           // Gets some subset of changes from a user profile.

           // Display the changes that have occurred in the last month
           DateTime dtNumberDays = DateTime.UtcNow.Subtract(TimeSpan.FromDays(30));
           // Create a change token to compare the number of days
           UserProfileChangeToken upChangeToken = new
                         UserProfileChangeToken(dtNumberDays);

           // Create a query object to determine which changes are displayed
           UserProfileChangeQuery QueryAnniversayAndColleagues = new
                         UserProfileChangeQuery(false,true);
           QueryAnniversayAndColleagues.ChangeTokenStart = upChangeToken;
           QueryAnniversayAndColleagues.Anniversary = true;
           QueryAnniversayAndColleagues.Colleague = true;

           string strUserName = "heididev\\brendon";
           if (pmManager.UserExists(strUserName))
           {
              UserProfile spMyUser = pmManager.GetUserProfile(strUserName);

              UserProfileChangeCollection MyUserProfileChanges =
                          spMyUser.GetChanges(QueryAnniversayAndColleagues);

              foreach (UserProfileChange MyUserChange in MyUserProfileChanges)
              {
                 Console.WriteLine(MyUserChange.EventTime.ToString());
                 if (MyUserChange is UserProfileAnniversaryChange)
                 {
                    UserProfileAnniversaryChange AnniversryEvent =
                          (UserProfileAnniversaryChange)MyUserChange;
                    Console.WriteLine(AnniversryEvent.Anniversary);
                    Console.WriteLine(AnniversryEvent.ProfileProperty);
                 }
                 else if (MyUserChange is UserProfileColleagueChange)
                 {
                    UserProfileColleagueChange ColleagueEvent =
                           (UserProfileColleagueChange)MyUserChange;
                 }
              }
           }
        }
     }
     catch (Exception exp)
     {
        Console.WriteLine(exp.Message);
     }
```

Building My Sites

My Sites are personalized sites that can be set up in SharePoint for and by each user individually. Each site can be provisioned on an as-needed basis to save disk space. It's similar to having self-serving Windows SharePoint Services sites, each set up for a different user. Although it is similar, SharePoint has added some features of My Site that make it useful for individual portals. Every site is different, based on the personalized settings that each user sets up for their own site. My Sites are central locations where users can store and manage information and documents that are specific to the user, such as Excel workbooks used to maintain sales for that person's region. Types of information that can be stored and managed on a user's My Site include: documents, tasks, calendars, links, and contacts. It is also a good way for other users in your organization to look up information about a user, including his responsibilities, skills, interests, and even current projects that he is working on.

My Sites have three basic pages that can be used for different purposes throughout the site:

- Public Profile Page
- Personal Site Page
- Personalization Page

Basic templates are provided for each type of page to give users a place to start. Also, each My Site page has a bar of links at the top of the page that can be used for easy navigation between these three different types of sites.

The Public Profile Page

The first page is the public profile page, called My Profile, which can be viewed by anyone in the organization. The administrator gets to decide how the page will look, but the users can add as much or as little information as they want. Privacy groups can be set up to let users decide who can view some of the information on their public site. Items such as skills, interests, distribution list memberships, a list of all colleagues, and other information can be set to only be seen by certain groups, such as a user's manager or workgroup members. This places some of the control of sensitive data in the user's hands and gives each user the ability to show only the information about themselves that they feel comfortable sharing.

The Personal Site Page

The second page that can be used is a personal site, and it is called My Home. Naturally, users access this site by clicking on My Home on the top link bar of My Site. The users are their own administrators for their personal site, and they are provided with a home page that only they can view. Since this page can only be viewed by the user, he or she is able to choose the look and layout of the site. Also, as administrators, users can create and manage as many pages as they want to for their site. With the exception of the private home page, personal site pages are set up to be shared with other selected people in the organization. Personal sites allow many users to work together on projects and store documents that can be shared by setting up list, document library, and workspace pages in their personal site.

The Personalization Page

The third page that is used is the personalization site, which is managed by the site administrator. It has the look and feel of the main site, except that changes have been made to identify it as another My Site page. There are also navigation sections at the top, and then four Web Part zones below. The zones, made up of the top, middle left, middle right, and bottom can be personalized. The content in each of them can be moved, deleted, or have new items added. However, these personalization sites can only be provided and changed by administrators or by content providers. The personalization site stores information that is personalized for each user, and allows special targeting to the user. Different groups in the organization can use this information to create sites that are personalized for each user, depending on their needs. This is discussed in more detail later in this chapter under “SharePoint Audiences.”

This type of page is very useful for presenting content that is personal to each user, and could not be posted publicly. For example, a manager could set up a personalization site that keeps track of annual reviews. When a user clicks on the site, all of her previous reviews can be seen, but she would only be able to view her own reviews. All of the personalization sites for each person are linked together to create a collection of sites, and there is a personalized navigation bar that gives the user easy access to their information.

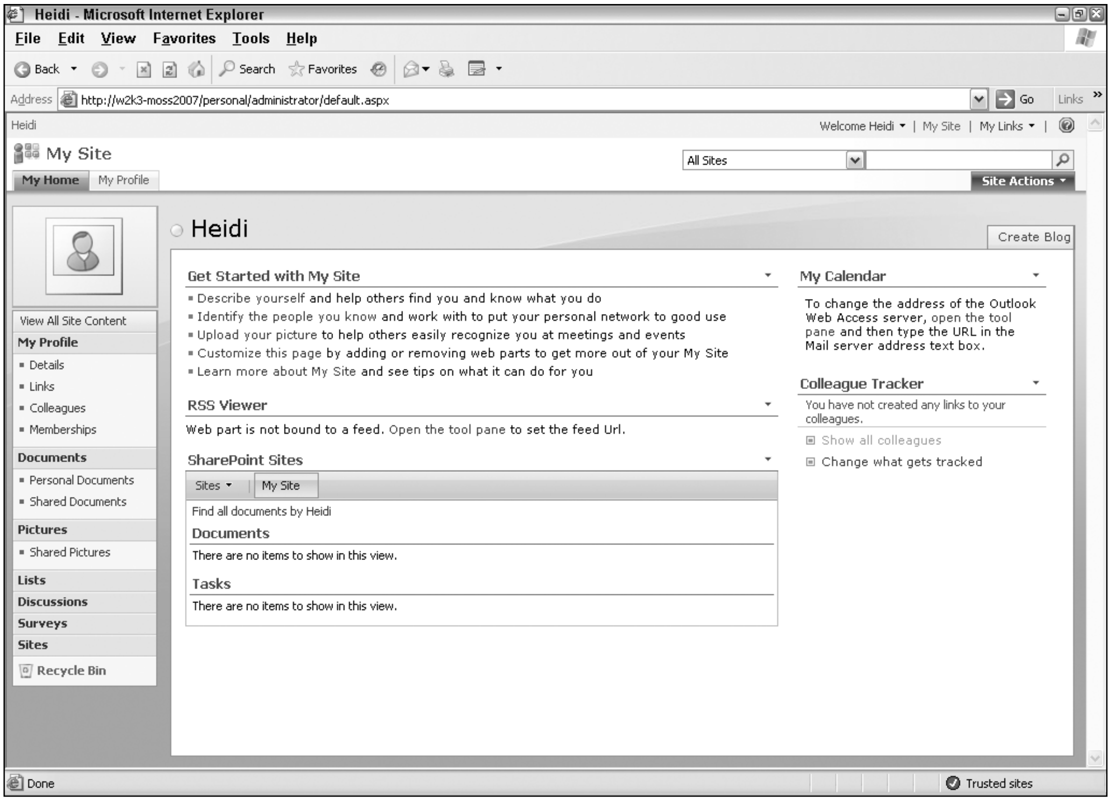
Creating Your First My Site

To create your first My Site page, follow these steps:

1. On the top right of the main screen, click My Site to create your My Site as the logged-in user. The very first time that a logged-in user clicks on the My Site link, SharePoint will create the site. You have to wait a few seconds while SharePoint sets up your personal site for the first time and creates the default document libraries and lists.
2. The first prompt that appears asks if you would like to have a quick link icon added to your personal site. This quick link icon will appear on the left of the Open and Save As dialog boxes inside of your windows applications. If you click Yes, then your personal site will be the first to appear in the drop-down list of locations when you create Meeting and Document Workspace sites. This also sets your My Site page to the default personal site. After SharePoint has set up your My Site, it is up to you to personalize it. Figure 8-3 shows the setup page you access to begin creating your own My Site page.

When personalizing My Site, information from the user profile will be displayed in the Public profile page when set to a visible Privacy setting. This allows for personalization based on the preferences of the user.

3. You can also keep a list of all of your favorite web sites so that you can access them from any computer where you have access to the network and are able to open My Site. At the top right of all pages on My Site, there is a link to the My Links page (this page can also be accessed by clicking on Links under My Profile on the Quick Launch). The My Links page lets you manage all of your links, as well as allowing access to sites where you are a member.



- Describe yourself and help others find you and know what you do
- Identify the people you know and work with to put your personal network to good use
- Upload your picture to help others easily recognize you at meetings and events
- Customize this page by adding or removing web parts to get more out of your My Site
- Learn more about My Site and see tips on what it can do for you

Web part is not bound to a feed. Open the tool pane to set the feed Url.

Find all documents by Heidi

There are no items to show in this view.

There are no items to show in this view.

To change the address of the Outlook Web Access server, open the tool pane and then type the URL in the Mail server address text box.

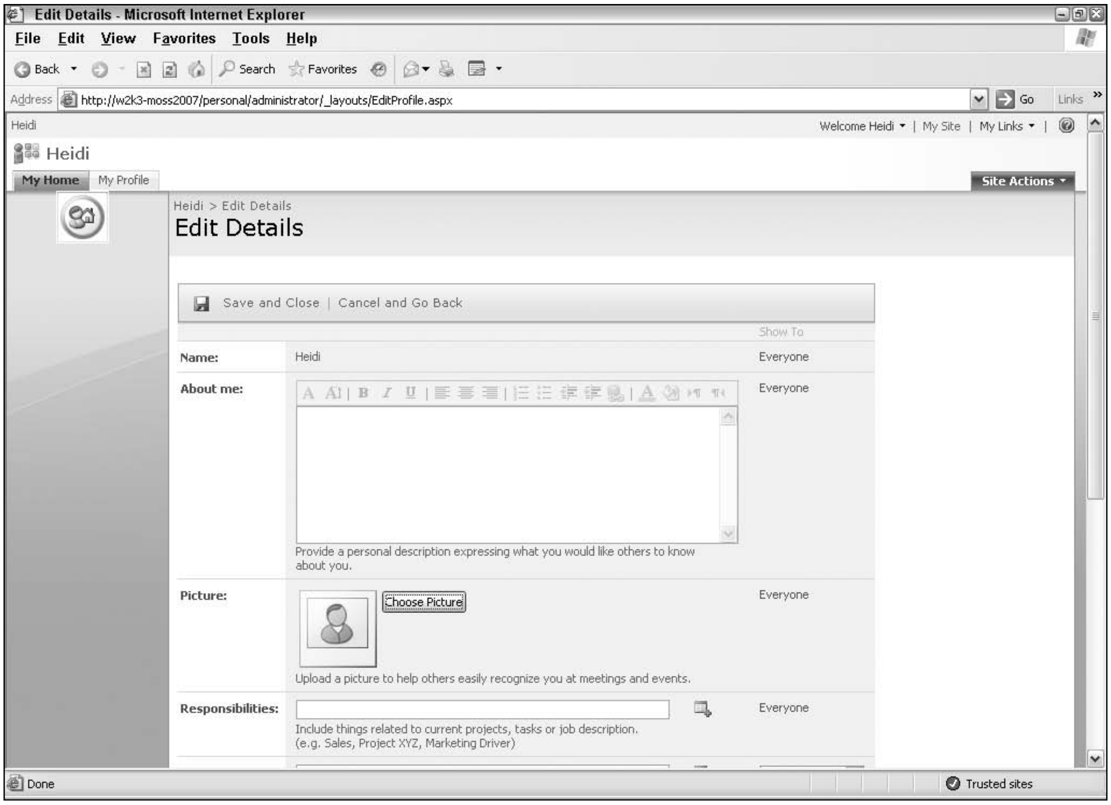
You have not created any links to your colleagues.

Figure 8-3

4. Once the profile has been imported or created a logged-in user can navigate to their My Site. Click on the link to "describe yourself," and there each available profile can be filled in with details such as a brief personal description, a photo, and the skills and responsibilities that you possess. As you can see in Figure 8-4, some properties can be edited and some cannot. These properties can also be modified using the object model.

5. Depending on how many user profile properties are configured on the site, there could be a lot of information that can be filled in, to give other people an inside look into each user. These profile properties can be modified or extended through the Shared Service Provider. This type of information sharing can also create an integrated organization, where people can see information about other users and relationships they have to others users with a customized set of data for each user. In fact SharePoint will monitor this data and make connections to other users such as colleagues.

   To the right of the fields is a drop-down box to select who the field will be shown to. Each field can be set to its own different value — such as Only Me, My Manager, My Workgroup, My Colleagues, and Everyone. This way, when logged in, you can fill something out but choose to only show it to certain groups. There are also a few fields that are set to Everyone and cannot be changed. When all changes have been made, click Save and Close at the top of the page.


Edit Details - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://w2k3-moss2007/personal/administrator/_layouts/EditProfile.aspx
Go Links
Heidi
Welcome Heidi | My Site | My Links
Heidi
My Home My Profile
Site Actions
Heidi > Edit Details
Edit Details
Save and Close | Cancel and Go Back
Choose Picture
Done
Trusted sites

		Show To
Name:	Heidi	Everyone
About me:	Provide a personal description expressing what you would like others to know about you.	Everyone
Picture:	Upload a picture to help others easily recognize you at meetings and events.	Everyone
Responsibilities:	Include things related to current projects, tasks or job description. (e.g. Sales, Project XYZ, Marketing Driver)	Everyone

Figure 8-4

These security settings for visibility are known as the policy settings. In the Shared Service Provider, each field can be set to a default privacy setting as well as have a value set indicating if the policy is Required, Optional, or Disabled. In addition to this setting, the site administrator can determine if the user can change the default privacy setting or not.

Identifying People You Know

The second link, Identify the people you know, lets the logged-in user identify people they know and work with. This creates a membership in SharePoint known as Colleagues. This helps you organize and manage your relationships to other users in the site. When users click Identify the people you know or the Colleagues link under the My Profile in My Site, a page where you can add colleagues and organize them into groups will be displayed. The users that are added will appear in the Colleague Tracker Web Part as well as any changes that a user has performed. Because the Colleagues are part of the `PrivacyPolicy` items, you have the option to set the visibility of the colleagues added to a profile.

Uploading Your Picture

To visually help identify a user, each profile has an associated picture. Clicking the Upload your picture link allows end users to modify their profile by displaying the Edit Details page used for the rest of the profile details. The profile property `Picture` allows end users to add a picture of themselves. This property is set to a field type of image and can be viewed by everyone. Fields can be of different types and can even be custom fields created through code.

Using a picture can be useful in large and small organizations, so that people can identify other people. In a large organization that may have many branches in different geographic locations, it could help someone put a face with a name of someone they talk to on the phone. In a small organization, it could help out new employees that have to meet a lot of new people all at once. If you add a picture to your profile, it will be shown on the upper-left corner of your My Site, and the file will also be listed in the Documents section.

Each My Site is just a site that is created for each user. All information and data is stored in a list on each site, so make sure to monitor the size of the My Site collection

Customizing Your My Site Page

Just like any site page, the My Site portal page can be customized with the user interface and Web Parts. Click the Customize this page link, and the page is shown in Edit Mode. The screen is split into zones, and My Site can be arranged in any way that is best for the user. The Get Started editing links can be taken away or moved to the bottom of the screen after setup, when they won't be used as much anymore. If you frequently use the My Calendar feature, it can be moved to the very top, so that it is the first thing you see on My Site. If there is a feature that is never used, for example if there are no RSS feeds set up, then that feature can be deleted from the page. Later on, if the user wants to add an RSS feed, they can add the Web Part back in by coming to this page and clicking on one of the Add a Web Part links in whatever zone they want it to show up in.

Using Help For End Users

SharePoint makes it easy to find out more information about the topic you are working with, and My Site is no different.

Click the Learn more about My Site link, which offers helpful tips. You can direct the end users to this location so that they can spend some time reviewing the information, as it can help them better understand My Site and how to use it. It will also introduce them to new uses for My Site, which will make it a more powerful tool for you and the organization.

Maintaining a Secure My Site

Privacy policies are built into SharePoint to protect the user from displaying information to users who should not be able to the view the data. When selecting data, there are many options that can be used as listed in the following Privacy enumerations. The end user can modify these settings on the profile page, while you as the developer will programmatically update these settings via the `Profile` property.

- **Public** — Everyone
- **Contacts** — Colleagues
- **Organization** — Workgroup members
- **Manager** — Manager

- **Private** — Only the user
- **NotSet —** Provides a default privacy setting

These restrictions apply to most of the My Site functionality.

The `UserProfileManager` class uses the method `GetPrivacyPolicy` to return the `PrivacyPolicyManager` for the current user. This manager will allow you to update these profile properties through code. To see how the `PrivacyPolicyManger` works, take a look at the following example:

1. Start by creating a console application named `PrivacyPolicyApp`.
2. Add the required SharePoint references:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Get the site and `UserProfileManager` references:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

4. Return the `PrivacyPolicyManager`, and loop through each `PolicyItem` in the site displaying relevant information about each item:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "wrox\\brendon";

      PrivacyPolicyManager ppmPolicyManager = pmManager.GetPrivacyPolicy();

      foreach(PrivacyPolicyItem tempItem in ppmPolicyManager.GetAllItems())
      {
         Console.WriteLine(tempItem.DisplayName);
         Console.WriteLine(tempItem.DefaultPrivacy);
         Console.WriteLine(tempItem.PrivacyPolicy);
         Console.WriteLine(tempItem.Group);
      }
      }
   }
   catch (Exception exp)
   {
      Console.WriteLine(exp.Message);
   }
```

The following code looks at the actual properties of a user and sets some of the privacy profile settings:

1. Start by creating a console application named `UserPrivacyPolicyApp`.
2. Add the required SharePoint references:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Get the site and `UserProfileManager` references:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

4. Get the reference to the user set in code and get the collection of properties:

```
string strUserName = "heididev\\brendon";

if (pmManager.UserExists(strUserName))
{
    UserProfile spUser = pmManager.GetUserProfile(strUserName);
    PropertyCollection pcCollection = pmManager.Properties;
}
```

5. From the properties collection get a property named `MLSNumber`. You will want to add this property from the Shared Service Provider if it does not exist.

```
Property property = pcCollection.GetPropertyByName("MLSNumber");
```

6. Set the privacy setting for the field to only be seen by you, the Manager:

```
property.DefaultPrivacy = Privacy.Manager;
```

7. Set the field to be a mandatory field that must be entered:

```
property.PrivacyPolicy = PrivacyPolicy.Mandatory;
```

8. Remember to always call the `Commit` function to save your changes in SharePoint:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "heididev\\brendon";

      if (pmManager.UserExists(strUserName))
      {
         UserProfile spUser = pmManager.GetUserProfile(strUserName);
         PropertyCollection pcCollection = pmManager.Properties;
         Property property = pcCollection.GetPropertyByName("MLSNumber");
         property.DefaultPrivacy = Privacy.Manager;
         property.PrivacyPolicy = PrivacyPolicy.Mandatory;
         property.Commit();
      }
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

Programmatically Using My Site

My Site is full of information about a user, and most of the relevant information is available through the SharePoint object model. Common actions that you can perform are to get My Site links as well as the user's My Site location. The next section looks at how to programmatically get this information.

Creating A My Site for Users

Some users in your system may need to have a My Site set up for them before they click on the My Site link themselves. This could happen when you are setting up users who store documents securely on their My Site page but mainly save those documents through rich client tools like Microsoft Office 2007 or a third-party application that can communicate with SharePoint 2007. When creating a My Site for a user, SharePoint provides a method called `CreatePersonalSite`, which is part of the `UserProfile` class. Most of the methods and properties used for creating My Site functionality are based on a single user created from the `UserProfile` class. This is due to the fact that the information is created from the user's profile and relationships.

To create the site, start by getting a reference to a new user or an already existing user, then call the `CreatePersonalSite()` method to create the My Site. These two steps are shown in the following code, where you perform the basic steps to get the profile manager and then make a call to the `CreatePersonalSite()` for the provided user.

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "wrox\\brendon";
      if (pmManager.UserExists(strUserName))
      {
         UserProfile spUser = pmManager.GetUserProfile(strUserName);
         Console.WriteLine("This may take a few minutes...");
         spUser.CreatePersonalSite();
      }
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

Creating a new site can take up to a few minutes, depending on how fast the server is and how much load the server is under at the moment. Make sure that you let the user know that his My Site is being provisioned and that it could take a few minutes.

Remember that if the My Site has already been created, it will not overwrite the My Site that is available. Instead, an exception will be thrown stating that the user already has a personal site created, and it will show the personal site's address.

Getting My Site Personal Information

In addition to providing methods for working with the My Site, the `UserProfile` class can also return information about the user's personalized data that makes up the My Site. In fact, with the object model that the Microsoft SharePoint team provides, you could create your own My Site or even add to the existing one.

The My Site that is provisioned is just a regular SharePoint site and can be modified with the SharePoint object model once you get a reference to the current site.

The following table shows some of the common personalized information provided in the `UserProfile` class that helps you view or update data:

Name	**Description**
`PersonalizationLinks`	Returns the set of links used for the Personalization links bar.
`PersonalSite`	Returns an object of type `SPSite` that represents the My Site of the current user.
`PersonalUrl`	Gets the URL for the My Site of the user that can be modified when logged in as the current user.

Continued

Name	Description
PublicUrl	Gets the URL of the current users My Site that can be viewed by any user. The sections marked as being viewable in the public profile are available.
QuickLinks	Gets the QuickLinkManager object for this user profile.

The PersonalSite property can be used to prevent the users My Site from being created twice. This property uses the user profile information to determine whether the My Site has been created already. The following code demonstrates how you might check to see if the user already has a My Site:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "wrox\\brendon";
      if (pmManager.UserExists(strUserName))
      {
         UserProfile spUser = pmManager.GetUserProfile(strUserName);
         if (spUser.PersonalSite == null)
         {
            Console.WriteLine("This may take a few minutes...");
            spUser.CreatePersonalSite();
         }
         else
         {
            SPSite personalSite = spUser.PersonalSite;
            Console.WriteLine("personalSite.Url: " + personalSite.Url);
         }
      }
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

To view the locations of the My Site for a user, locate or type the name of the user that you need the information for, and then display either the PersonalUrl or the PublicUrl. The personal URL displays the location of the site that the user can log in to and edit. If the site is not associated with the currently logged-in user, the page will redirect the user to the View All Content page, but if it is the personal My Site of the logged-in user, My Site will appear and can be edited by the user.

When the user has a My Site already created, the PersonalUrl and the URL of the SPSite returned in the PersonalSite will be the same. If the user has not already set up a My Site, the personal and public URLs that are returned are also the same. This is due to the fact that the personal URL is updated when the My Site is actually created.

Sometimes, you will want to provide a link to the My Site of a user for other users to view. In that case, use the `PublicUrl` of the user's My Site to display the URL that any user can see. This page can show information that has been correctly set up in the privacy policies and will display information according to which user you are logged in as. The following code displays the URL information of the current user to the console application. The URLs that are displayed are the `PersonalUrl` and `PublicUrl` of the current user.

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "wrox\\brendon";
      if (pmManager.UserExists(strUserName))
      {
         UserProfile spUser = pmManager.GetUserProfile(strUserName);
         Console.WriteLine("spUser.PersonalUrl");
         Console.WriteLine(spUser.PersonalUrl);
         Console.WriteLine("spUser.PublicUrl");
         Console.WriteLine(spUser.PublicUrl);
      }
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

In addition to retrieving the URL properties and SharePoint site of the My Site, you can also work with the user's links. You can use either the `QuickLinks` or the `PersonalizationLinks` from the user's profile. If you select `GetItems()` without providing a privacy policy enumeration, then the default will be `Privacy.Private`. You may want to add a link to every user's profile or just display the user's link on another Web Part. The following code shows how to get a user's links and display them in the console window:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "heididev\\sam";
      if (pmManager.UserExists(strUserName))
      {
         UserProfile spMyUser = pmManager.GetUserProfile(strUserName);
         QuickLinkManager linkManager = spMyUser.QuickLinks;
         foreach (QuickLink link in linkManager.GetItems())
         {
            Console.WriteLine(link.Title + " : " + link.Url);
         }
      }
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

Colleagues

You can use Colleagues to gather information about related contacts in SharePoint. Colleagues are other users in the system that are your managers, peers, or direct reports. These users are automatically added as colleagues by SharePoint 2007. You might have colleagues that are part of your organization that do not fit into one of these categories. This could be a person in another department, who is not part of your reporting structure. You can add these users as colleagues who will appear in the My Site page. Figure 8-5 shows the Colleagues page. Here you can add, and manage colleagues and view the relationships among colleagues.

Once the colleagues have been added, you can display them with the Colleague Tracker Web Part, which is loaded with Microsoft Office SharePoint Server 2007. By default, each user's My Site page contains a Colleague Tracker Web Part to easily view the colleagues each user has. Colleagues can also be used to narrow the searches that are performed on the site.

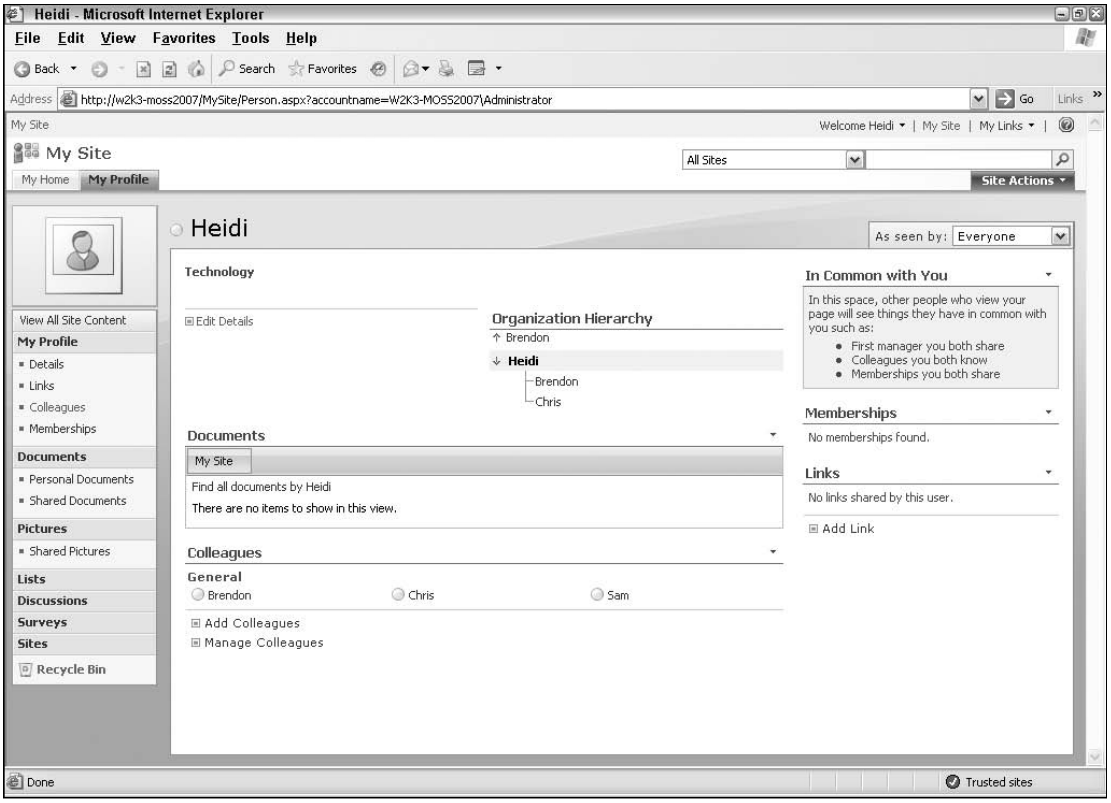


Figure 8-5

The UserProfile class contains a property named Colleagues that returns the colleague's manager for the current user, who can be used to work with the colleagues of the user. Once you have a reference to the ColleagueManager you can add, remove, or list the colleagues of the user. There are even built-in methods to list the common colleagues as well as colleagues of colleagues. The following table lists the ColleagueManager methods for managing a user's colleagues:

Name	Description
Create	Creates a Colleague object with the specified type and privacy level
DeleteAll	Removes all of the colleagues associated with the user
GetColleaguesOfColleagues	Gets the set of colleague's colleagues
GetCommonColleagues	Gets the colleagues that the user and the colleague have in common or share
GetItems	Returns the set of all colleagues based on the privacy settings
IsColleague	Determines whether the specified user is a colleague of this colleague

To add a new colleague, use the built-in method, IsColleague, to check whether the user is already a colleague. If the user is not a colleague, then make a call to the Create method, as follows:

1. Start by creating a console application named ColleagueManagerApp.
2. Add the required SharePoint references:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Get a reference to the ColleagueManager from the UserProfile object:

```
 spMyUser = pmManager.GetUserProfile(strUserName);
ColleagueManager cmColleagues = spMyUser.Colleagues;
```

4. Display the number of colleagues that have a privacy setting of Public:

```
int NumColleages = cmColleagues.GetItems(Privacy.Public).GetLength(0);
Console.WriteLine(NumColleages);
```

5. Check to see if the provided user is a colleague of the first user. If the user is not a colleague, add the user as a colleague:

```
string strNewColleague = "heididev\\sam";
UserProfile spMyColleague = pmManager.GetUserProfile(strNewColleague);
```

```
if (!cmColleagues.IsColleague(spMyColleague.ID))
{
       cmColleagues.Create(spMyColleague,
                           ColleagueGroupType.Peer,
                           "Peers",
                           false,
                           Privacy.Public);
}
```

6. Loop through each colleague and display them to the console window:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "heididev\\brendon";
      if (pmManager.UserExists(strUserName))
      {
         UserProfile spMyUser = pmManager.GetUserProfile(strUserName);
         ColleagueManager cmColleagues = spMyUser.Colleagues;

         int NumColleages = cmColleagues.GetItems(Privacy.Public).GetLength(0);
         Console.WriteLine(NumColleages);

         string strNewColleague = "heididev\\sam";
         UserProfile spMyColleague = pmManager.GetUserProfile(strNewColleague);
         if (!cmColleagues.IsColleague(spMyColleague.ID))
         {
            cmColleagues.Create(spMyColleague,
                                ColleagueGroupType.Peer,
                                "Peers",
                                false,
                                Privacy.Public);
         }

         foreach (Colleague tempColleague in cmColleagues.GetItems(Privacy.Public))
         {
            Console.WriteLine(tempColleague.Title);
            Console.WriteLine(tempColleague.GroupType);
            Console.WriteLine(tempColleague.IsAssistant);
            Console.WriteLine(tempColleague.IsInWorkGroup);
         }
      }
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

By using the `ColleagueManager` class, you can remove all of the colleagues of a single user. However, if you want to delete just one colleague, you need to get a reference to that colleague and then call the `Delete` method from the `Colleague` class.

Membership

Similarly to colleagues, memberships provide relationships between the user and content. Content can be related to the user through automatic association or you can use the object model. Standard functionality includes such things as determining memberships, providing the ability to create new memberships, and displaying information about current memberships.

There are two types of membership:

- Distribution List (DL) memberships
- Microsoft Windows SharePoint Services 3.0 site memberships

The difference between the two types of memberships is that one is created from the Active Directory directory service and the other is created from SharePoint sites.

Again, just like the colleagues, the memberships that a user has are displayed on the My Site. When another user views the common page, also known as the Profile page, the common memberships with that user will also be shown.

There are two sets of classes to use when programming with the membership relationships that allow you to work with the membership groups and the memberships themselves. The classes are:

- `MembershipGroup`
- `Membership`

Both of these classes have a manager class that is associated with them to manage the collections of these items. The manager class allows you to add, display or change the membership objects that they contain.

Working with Membership Groups

You will use the membership groups to add users as a member of a distribution list or to a SharePoint site membership. The membership groups use a number of public properties to describe the membership group itself. The following table lists some useful properties of the class:

Name	**Description**
`CreateMemberGroup`	Allows developers to create members groups from the object model
`Count`	Returns the number of member groups that are associated with the user
`GetMemberGroupBySourceAndSourceReference`	Returns a specific `MemberGroup` based on the source and name of the source reference

The following code uses the `UserProfile` class to get the membership groups that the user has and then display the number of memberships the user has. In addition to showing the values of the `Membership` class, the `MembershipGroupManager` is used to get the list of groups:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "wrox\\brendon";
      if (pmManager.UserExists(strUserName))
      {
         UserProfile spMyUser = pmManager.GetUserProfile(strUserName);
         MemberGroupManager mgmMemberGroups = pmManager.GetMemberGroups();

         int NumMemberships = mgmMemberGroups.Count;
         Console.WriteLine("Number of Memberships: " + NumMemberships);
      }
   }
}
catch (Exception exp)
{
   Console.WriteLine(exp.Message);
}
```

If the installation of SharePoint is new, no membership groups may have been created yet. To add a new group, use the `MemberGroupManager` and create a new group with the `Create` method. Now that you have a `MemberGroup`, you can display the group with the properties of the `MemberGroup` class, which are discussed in the following table:

Name	**Description**
`Count`	Returns the number of members associated the member group
`Description`	A description of the text describing the member group
`DisplayName`	Name used when the member group is displayed in the user interface
`Source`	The GUID of the source member group
`SourceReference`	The value that makes the member group unique in the source
`Url`	The location of the membership group's URL

When creating new groups, the `sourceReference` string must be different for each `MembershipGroup` you add, or they must have a different source location. An error will be displayed if you try to add the same source more than once. You can however have the same `sourceReference` name if it's from different source. A source must be a valid GUID that can be user created, or it can be one of the predefined types. The following sources are part of the `PrivacyPolicyIDConstants` enumeration class:

- ❑ `MembershipsFromDistributionLists`
- ❑ `MembershipsFromSharePointSites`
- ❑ `MyColleaguesOnMySite`
- ❑ `MyColleaguesRecommendations`
- ❑ `MyPersonalizationLinksOnMySite`

1. Start by creating a console application named `MembershipManagerApp`.
2. Add the required SharePoint references:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Get the site and `UserProfileManger` reference.
4. Get the list of `MemberGroups` for the user; if there are zero groups create a new one. Make this new list a `DistributionList`.

```
if (mgmMemberGroups.Count == 0)
{
       // Create new member group
       MemberGroup newGroup = mgmMemberGroups.CreateMemberGroup(
                    PrivacyPolicyIdConstants.MembershipsFromDistributionLists,
                    "Wrox SP Reviewers List",
                    "Wrox Review",
                    "Reviewers list",
                    "http://localhost",
                    "Wrox");
}
```

5. Display the number of groups that are in the list:

```
        int NumMemberships = mgmMemberGroups.Count;
        Console.WriteLine("Number of Memberships: " + NumMemberships);
```

6. Loop through each member group and display it to the console window:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
```

```
            string strUserName = "wrox\\brendon";
            if (pmManager.UserExists(strUserName))
            {
               UserProfile spMyUser = pmManager.GetUserProfile(strUserName);
               MemberGroupManager mgmMemberGroups = pmManager.GetMemberGroups();

               if (mgmMemberGroups.Count == 0)
               {
                  // Create new member group
                  MemberGroup newGroup = mgmMemberGroups.CreateMemberGroup(
                               PrivacyPolicyIdConstants.MembershipsFromDistributionLists,
                               "Wrox SP Reviewers List",
                               "Wrox Review",
                               "Reviewers list",
                               "http://localhost",
                               "Wrox");
              }

              int NumMemberships = mgmMemberGroups.Count;
              Console.WriteLine("Number of Memberships: " + NumMemberships);

              foreach (MemberGroup tempMemberGroup in mgmMemberGroups)
              {
                 Console.WriteLine(tempMemberGroup.DisplayName);
                 Console.WriteLine(tempMemberGroup.Description);
                 Console.WriteLine(tempMemberGroup.MailNickName);
                 Console.WriteLine(tempMemberGroup.Source);
                 Console.WriteLine(tempMemberGroup.SourceReference);
              }
           }
        }
    }
    catch (Exception exp)
    {
       Console.WriteLine(exp.Message);
    }
```

Now that you have membership groups, you can create a membership relation to one of the users in your SharePoint system. Use the `GetMemberGroupBySourceAndSourceReference()` method of the `MemberGroupManager` class to get a reference to the `MemberGroup`. Make sure to use the correct source and `sourceReference` to return the right group. If you create your own source, make sure to store it in a permanent location such as a database to be able to retrieve it for use with the `Get` statements. You can also iterate through each of the items until you find the right one. Once you have the reference to the user and the membership groups, call the `Create` method of the `MembershipManager` class.

The membership item has the same privacy policy as other items in the My Site. When you set the membership relationship, be sure to provide the right level of privacy. This also applies to the `GetItems` method of the membership manager, which is used to retrieve the `Membership` objects. The default privacy policy is `Private`, so make sure to specify a different level if required. The following code shows how to get the membership groups of a user and then check for a specific group:

1. Start by creating a console application named `MembershipManagerApp`.

2. Add the required SharePoint references:

```
using System;
using System.Collections.Generic;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.UserProfiles;
```

3. Get the site and `UserProfileManger` reference.
4. Return the `DistributionLists` with a reference of Wrox for the current user.

```
MemberGroup mgMemberGroup =
          mgmMemberGroups.GetMemberGroupBySourceAndSourceReference(
                  PrivacyPolicyIdConstants.MembershipsFromDistributionLists,
                  "Wrox");
```

5. Check to see if the user already has a membership; if the user doesn't, create a new membership to the group:

```
MembershipManager mmMemberships = spMyUser.Memberships;

//Create a new membership with the current user
if (mmMemberships[mgMemberGroup] == null)
{
      mmMemberships.Create(mgMemberGroup,
                           MembershipGroupType.UserSpecified,
                           "Wrox Reviewers",
                           Privacy.Public);
}
```

6. Display all the memberships the user has to the console window:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      UserProfileManager pmManager = new UserProfileManager(siteContext);
      string strUserName = "wrox\\brendon";
      if (pmManager.UserExists(strUserName))
      {
         UserProfile spMyUser = pmManager.GetUserProfile(strUserName);
         MemberGroupManager mgmMemberGroups = pmManager.GetMemberGroups();

         MemberGroup mgMemberGroup =
               mgmMemberGroups.GetMemberGroupBySourceAndSourceReference(
                       PrivacyPolicyIdConstants.MembershipsFromDistributionLists,
                       "Wrox");

         MembershipManager mmMemberships = spMyUser.Memberships;

         //Create a new membership with the current user
```

```
            if (mmMemberships[mgMemberGroup] == null)
            {
              mmMemberships.Create(mgMemberGroup,
                                      MembershipGroupType.UserSpecified,
                                      "Wrox Reviewers",
                                      Privacy.Public);
            }

            foreach (Membership tempMembership in mmMemberships.GetItems(Privacy.Public))
            {
               Console.WriteLine(tempMembership.GroupType);
               Console.WriteLine(tempMembership.Title);
               Console.WriteLine(tempMembership.Url);
               Console.WriteLine(tempMembership.MembershipGroup.MailNickName);
            }
         }
      }
   }
   catch (Exception exp)
   {
      Console.WriteLine(exp.Message);
   }
```

SharePoint Audiences

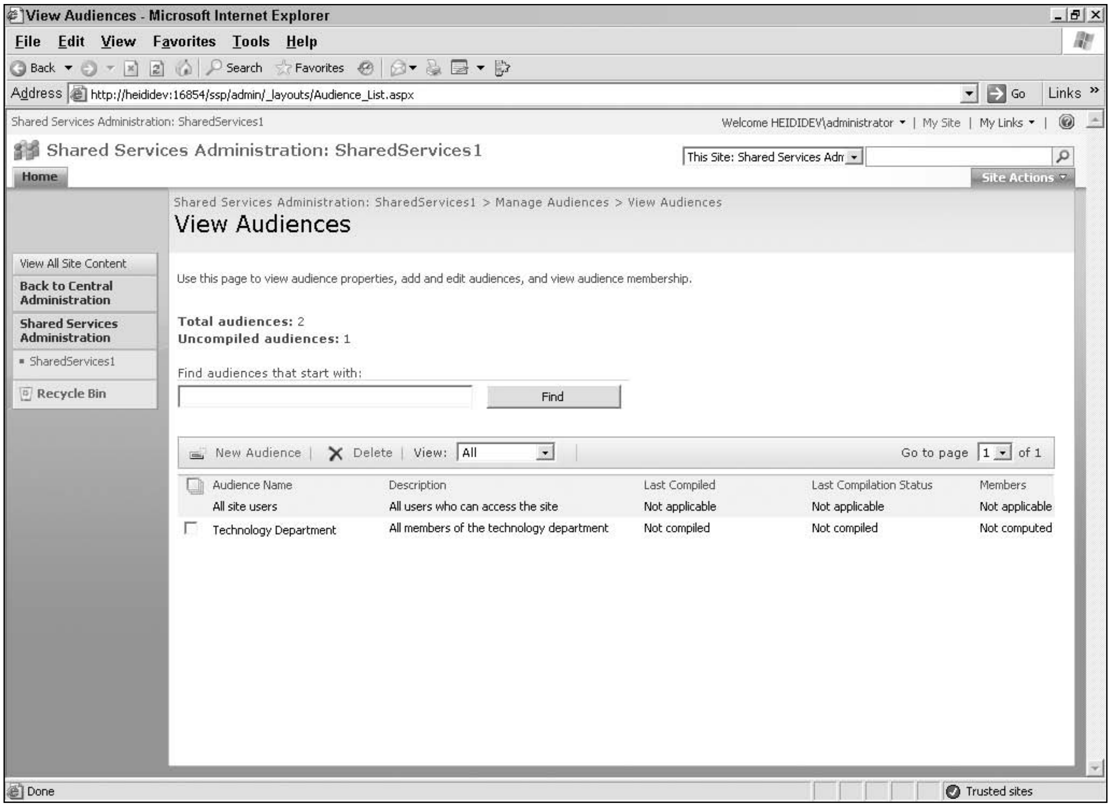
SharePoint allows content providers to customize information by using audience targeting through a rules based system. An audience is a collection of users that someone wants to target. Audiences can be groups of people from the Windows security groups, or they can be groups that include people based on rules you set up. They are able to do this by using personalization sites to target users based on information in their user profiles. For example, an audience can be created that includes all members of the accounting department. Once that audience is created, it can then be used to target members of that audience in order to provide custom content to them. Pages and listings can be targeted to an audience to enhance the user experience and provide meaningful custom content. Figure 8-6 shows the list of audiences as they would appear in the Shared Service Provider.

In SharePoint 2007 you can perform audience targeting on a rules-based audience, Windows SharePoint Services groups, and memberships.

Creating an Audience

Audiences can only be created and managed by the administrator from the Shared Services Administration page or by using the object model. To create an audience, follow these steps:

1. Inside the Shared Service Provider page click Audiences, and the Manage Audiences page comes up, which will allow you to manage and compile the audiences you have created.
2. To view one of the existing audiences, click the View Audiences link.
3. To create a new audience, click Create audience, and you are able to define the settings for the audience you would like to create. The first thing needed is a unique name that will easily identify the audience. A description can also be added, along with the owner of the audience. The last option is to include either users who meet all of the rules of the audience, or just any of the rules.


View Audiences - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://heididev:16854/ssp/admin/_layouts/Audience_List.aspx
Go Links
Shared Services Administration: SharedServices1
Welcome HEIDIDEV\administrator | My Site | My Links
Shared Services Administration: SharedServices1
This Site: Shared Services Adm
Home
Site Actions
View All Site Content
Back to Central Administration
Shared Services Administration
SharedServices1
Recycle Bin
Shared Services Administration: SharedServices1 > Manage Audiences > View Audiences
Find
New Audience | Delete | View: All
Go to page 1 of 1
Done
Trusted sites

View Audiences

Use this page to view audience properties, add and edit audiences, and view audience membership.

Total audiences: 2
Uncompiled audiences: 1

Find audiences that start with:

Audience Name	Description	Last Compiled	Last Compilation Status	Members
All site users	All users who can access the site	Not applicable	Not applicable	Not applicable
Technology Department	All members of the technology department	Not compiled	Not compiled	Not computed

Figure 8-6

Creating Audience Rules

Once an audience is created, you must define the rules that users must meet to be included in the audience. The rules you are able to define use simple queries that will either include or exclude users from the audience. When using the user interface to create rules governing who to include in your audience, you are limited to only six elements in your rules. If a rule has more than six elements that are used, then it must be created by using the object model.

When entering rules using the user interface, each rule asks you to define an operand, an operator, and a value. The operand is the user or the property that you are looking to include in the audience. This is either a user or a property. Select User when you want to add a rule that is based on distribution list membership, a reporting structure, or a security group. When User is chosen as the operand, the only available options for the operator are Reports Under and Member Of. The value must then be a user of the organization. Alternatively, choose a profile property when you want to choose one of the properties that are listed on the drop-down menu.

The operator is what determines how the operand is evaluated, and ultimately whether a user is included in the audience or not. The list of operators that are available to choose from will change, depending on the operand that was chosen. The value is the single value that you want to compare the operand to, using the rule of the chosen operator. The following example should help clarify the different parts used when defining a rule. If the operand is "First name", the operator is "Contains", and the value is "tom", then every user whose first name contains the letters "tom" will be included in the audience. However, it's important to note that this will include anyone by the name of Tom or Tommy, but not anyone with the name of Thomas.

Once you have added a rule, the View Audience Properties page is shown. There, the audience properties and rules are displayed for review. From this page, there are links to edit the audience properties, view the membership, and compile the audience. Also, under the Audience Rules section, the rules can be clicked on to edit them.

Audience Targeting

In the past, SharePoint Portal Server 2003 allowed audience targeting to be done based on a fairly small set of rules. However, Office SharePoint Server 2007 allows you to target an audience based on a lot more information, including groups, distribution list (DL) memberships, and rules-based audiences. Instead of making you target the entire list, SharePoint Server 2007 also lets you target content just by using a list item, making it much more efficient. This type of audience targeting is made possible by using filters on user profiles and by using personalized Web Parts to display content. There are no default Web Parts set up on a page, but each site utilizes different filter Web Parts that help determine which Web Parts on the page should be shown to the user who is visiting the site. It is also interesting to know that SharePoint Server 2007 can automatically find all of the sites that are targeted to a particular user.

Targeting Content

Once you have created an audience, you can then start targeting content for that audience. As an example, using your My Site page, on the Quick Launch, select the page you would like to use for audience targeting.

1. Click on the list or document library you would like to use, and the list of items will be displayed.
2. From there, click on the Settings tab, and choose List Settings or Document Settings to pull up all of the settings for that page.
3. Under General Settings, click on Audience targeting settings.
4. Check the Enable audience targeting check box, which will enable audience targeting for the chosen list or document library.

When audience targeting is enabled, a column is added to the list or document library. The data in this new column will be used by Web Parts to filter the list or library contents based on the user information provided. This enables the content in that Web Part to be customized to the specific user or audience. SharePoint Server 2007 gives you the ability to target items or links in a list to an audience by using Web Parts that are able to recognize membership in that audience. When this is done, only members of that particular audience are able to see the targeted items in that Web Part.

A great example of how this could be used is a human resources department at a large company. The department could use a News Web Part on their home page that would use audience targeting to deliver news items that are relevant to certain users based on audience targeting. For example, if a company wanted to distribute sensitive information that only applied to executive-level managers or higher, they could use audience targeting to accomplish this. By using this method, the new information would only be shown to users it applied to, and important messages could be delivered to executives in an efficient manner without going to lower-ranking employees.

After audience targeting has been enabled, the Settings page for the list or library you changed will show the newly created column. Under the list of all columns, there is a new one listed, the Target Audiences column. The column heading is Audience Targeting, but it is not required. If you click on the column, you will be taken to the Edit page. On this page, you can change the column name, add a description, and change whether or not information is required for this column. However, you are also given the option to allow a global audience, allow distribution lists, and allow SharePoint Groups. All of these options are initially checked but can be changed if you prefer to not allow them. There is also a button that will allow you to delete this column if it is no longer needed. If it is deleted, the next time you go back to the Audience Targeting, the Enable audience targeting check box will not be checked, and therefore audience targeting will not be enabled unless you enable it again.

1. After audience targeting has been enabled, go back to the List page and add a new list item by clicking New and then the item type.
2. On the New Item page, there is now a field to fill in for Target Audiences. You can now choose from the audiences that have been created to target this list item to.
3. Click on the browse icon to pull up a list of available audiences that have been created. Distribution lists and security groups can also be used in this field, but only after they have been imported. One option that can be chosen is All site users, which will allow everyone to see this list. If you are working with a document library where audience targeting has been enabled, audience targeting can be set when you edit or upload a document in the same way as discussed here.

Displaying Targeted Content

Once the audience targeting has been set up on the content, the web site must be changed to display it.

1. Log on to SharePoint using one of the accounts that can modify the Shared View.
2. Click on the link to edit the page, and the page will then be displayed with the Web Part zones.
3. Choose one of the zones, and click to add a Web Part.
4. Click on the check box in the Default group next to the Content Query Web Part to add a Content Query Web Part to the page. This will enable you to show the targeted content from your site to users when they pull up the web page.
5. When the Web Part has been added, stay in edit mode for the page, and click on the edit link on the Content Query Web Part. Choose to Modify Shared Web Part.
6. The Web Part settings page will show up on the right side of the page, where the Content Query Web Part can be modified.
7. Click on the plus sign next to the word Query to expand the Query topic section.
8. Change the List Type to match your audience targeting page, whether it's a specific list (like a tasks list or a links list) or a document library.
9. Next, check the apply audience filtering box under Audience Targeting.
10. Once these changes have been made, click on OK. Now, the list or document library that you selected will only show up for the users that have been targeted in the chosen audience.

Working with the Audiences API

When working with the Audiences API, SharePoint 2007 has introduced a new namespace implemented in `Microsoft.Office.Server.Audience`. The SharePoint team has left the previous SharePoint 2003 version to maintain backward compatibility, but you should work with the new namespace when creating new applications.

Code and namespaces that are marked as deprecated or obsolete may not function as expected in new versions of SharePoint. Make sure to test your code to make sure that everything still works as it did in SharePoint 2003.

The major use of audiences is to ensure that content is delivered to the right people. Although it may seem like this can be used to secure locations and documents, audience targeting is not used as a security mechanism in SharePoint 2007.

To pull the right information about users, audiences must be compiled based on the rules provided. When the rule is being compiled, SharePoint will use information from Active Directory as well as SharePoint itself and store the compiled data result into the database. When external data changes, SharePoint will not recompile the data automatically, and the audience must be recompiled once again.

As with the other objects you have seen, the audience API has a manager that is used as the top-level object when working with the site. The `AudienceManager` class uses the server context to retrieve the correct `AudienceManager` data. Take a look at the following code and notice that the syntax is very similar to the `UserProfile` store. One reason that this may look similar is because the audience system is part of the user profile system. The difference is just that the data is stored in a separate database table.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.Audience;

namespace Audiences
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                using (SPSite spSite = new SPSite(@"http://localhost"))
                {
                    ServerContext siteContext = ServerContext.GetContext(spSite);
                    AudienceManager amManager = new AudienceManager(siteContext);

                    if (!amManager.Audiences.CompileInProgress)
                    {
                        Console.WriteLine(amManager.Audiences.Count);
                    }
                }
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception.ToString());
            }
        }
    }
}
```

Adding New Audiences

When working with either a simple audience rule or a complex audience rule, you can create the audience with the `AudienceCollection` object that is returned from the Audience Manager.

The `Audience` class has a number of useful properties that can be displayed and objects that can be updated. These would be objects such as `Audience Name`, AudienceRules, and `AudienceDescription`. You will use these properties and objects in code shortly.

To create a simple audience, use the site elements server context to retrieve the `AudienceManager` class and the `AudienceCollection`. It is always a good idea to check if the audience already exists on the current site. If you try to add an audience twice, an exception will be thrown that notifies you that the duplicated name is already available for the current site.

1. Start by creating a console application named `AudienceApp`.
2. Add the required SharePoint references:

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.Audience;
```

3. Get the site and `AudienceManager` reference:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      AudienceManager amManager = new AudienceManager(siteContext);
   }
}
catch (Exception exception)
{
   Console.WriteLine(exception.Message);
}
```

4. Get a collection of all of the Audiences in the site:

```
AudienceCollection acCollection = amManager.Audiences;
```

5. Check if an audience named Marketing Department already exists, and if it doesn't, create the audience:

```
string strName = "Marketing Department";
string strDescription = "All members of the marketing department";

if (!acCollection.AudienceExist(strName))
{
   acCollection.Create(strName, strDescription);
}
```

6. Loop through each of the audiences and display the `AudienceName`, `AudienceSite`, and `MembershipCount`:

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      AudienceManager amManager = new AudienceManager(siteContext);

      AudienceCollection acCollection = amManager.Audiences;
      string strName = "Marketing Department";
      string strDescription = "All members of the marketing department";

      if (!acCollection.AudienceExist(strName))
      {
         acCollection.Create(strName, strDescription);
      }

      foreach (Audience tempAudience in acCollection)
      {
         Console.WriteLine(tempAudience.AudienceName);
         Console.WriteLine(tempAudience.AudienceSite);
         Console.WriteLine(tempAudience.MemberShipCount);
      }
   }
}
catch (Exception exception)
{
   Console.WriteLine(exception.Message);
}
```

Adding Rules to Your Audience

Rules help to determine who the targeted information will be displayed to. The rules are string-based values that are evaluated by an operator. Each rule contains a left operand value used for the `AudienceRule` class, an operator value provided when creating a new rule, and a value for comparing the operand. The standard operators AND/OR can be used when creating your rules. In addition to the AND/OR operators, you can use parentheses to create more complex rules. However, you are allowed a maximum of three levels of parentheses nesting.

Supported Audience Rules include the following:

- =
- >
- >=
- <
- <=
- Contains
- Reports Under
- <>
- Not contains
- AND
- OR
- (
-)
- Member of

When creating an audience, it is a good idea to check if the audience already has rules associated with the audience. If there are rules associated with the audience, you should create a new AND rule that will include both the old rules and the new rules you are adding.

Make sure to call the `Commit` method on the audience once you have added all of the rules to make the new rules take effect on the server.

1. Start by creating a console application named `AudienceApp`.
2. Add the required SharePoint references:

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.SharePoint;
using Microsoft.Office.Server;
using Microsoft.Office.Server.Audience;
```

3. Get the site and `AudienceManager` reference.
4. Get a collection of all of the audiences in the site:

```
AudienceCollection acCollection = amManager.Audiences;
```

5. Check to see if the audience already exists. If so, retrieve the audience from SharePoint, and if it doesn't, create the new one:

```
 Audience.string strName = "Technology Department";
string strDescription = "All members of the technology department";
Audience TechAudience = null;
if (acCollection.AudienceExist(strName))
{
     TechAudience = acCollection[strName];
}
else
{
     TechAudience = acCollection.Create(strName, strDescription);
}
```

6. Get the rules that are associated with the audience. If the audience already has rules associated with it, then create a new rule component that will AND the old rules and the new rules together. Otherwise, just create a new empty `ArrayList`.

```
ArrayList techRules = TechAudience.AudienceRules;
if (techRules == null)
{
    techRules = new ArrayList();
}
else
{
    techRules.Add(new AudienceRuleComponent(null, "AND", null));
}
```

7. Finally, add two new rules to the audience that will limit the people in the audience. These rules are also combined by the AND operator. You can use any of the operators seen in the list above, depending on the type of comparison you wish to apply.

```
try
{
   using (SPSite spSite = new SPSite(@"http://localhost"))
   {
      ServerContext siteContext = ServerContext.GetContext(spSite);
      AudienceManager amManager = new AudienceManager(siteContext);

      AudienceCollection acCollection = amManager.Audiences;

      string strName = "Technology Department";
      string strDescription = "All members of the technology department";
      Audience TechAudience = null;
      if (acCollection.AudienceExist(strName))
      {
         TechAudience = acCollection[strName];
      }
      else
      {
         TechAudience = acCollection.Create(strName, strDescription);
         }

         ArrayList techRules = TechAudience.AudienceRules;
         if (techRules == null)
         {
            techRules = new ArrayList();
         }
         else
         {
            techRules.Add(new AudienceRuleComponent(null, "AND", null));
         }

         AudienceRuleComponent arRule1 = new AudienceRuleComponent("FirstName",
   "Contains", "H");
         techRules.Add(arRule1) ;

         AudienceRuleComponent arRule2 = new AudienceRuleComponent(null, "AND", null);
         techRules.Add(arRule2);

         AudienceRuleComponent arRule3 = new AudienceRuleComponent("BrokerID", "=",
   "090101");
         techRules.Add(arRule3);

         TechAudience.AudienceRules = techRules;
         TechAudience.Commit();
      }
   }
   catch (Exception exception)
   {
      Console.WriteLine(exception.Message);
   }
```

Removing Audiences

You may need to manage the audiences by removing one of the existing audiences created from the Shared Service Provider Central Administration page or from your code. SharePoint makes it easy to remove the audience by using the GUID of the audience or the name of the audience.

In contrast to adding an audience, you should now make sure that the audience does exist, to prevent an exception from being thrown. The following code removes an audience called Technology Department:

```
   try
   {
      using (SPSite spSite = new SPSite(@"http://localhost"))
      {
         ServerContext siteContext = ServerContext.GetContext(spSite);
         AudienceManager amManager = new AudienceManager(siteContext);

         string strName = "Technology Department";

         if (amManager.Audiences.AudienceExist(strName))
         {
            amManager.Audiences.Remove(strName);
         }
      }
   }
   catch (Exception exception)
   {
      Console.WriteLine(exception.Message);
   }
```

Working with Audience Memberships

Most companies want to know which users are in each audience, so SharePoint has provided a way to get the memberships associated with an audience. Use the `Audience` class, and call the method `GetMembership` to return a list of memberships as shown in the following code:

```
   try
   {
      using (SPSite spSite = new SPSite(@"http://localhost"))
      {
         ServerContext siteContext = ServerContext.GetContext(spSite);
         AudienceManager amManager = new AudienceManager(siteContext);

         string strName = "Technology Department";
         Audience TechAudience = amManager.Audiences[strName];

         ArrayList TechAudienceMembers = TechAudience.GetMembership();

         foreach (UserInfo TechUser in TechAudienceMembers)
         {
            Console.WriteLine(TechUser.PreferredName);
            Console.WriteLine(TechUser.NTName);
            Console.WriteLine(TechUser.Email);
         }
      }
   }
   catch (Exception exception)
   {
      Console.WriteLine(exception.Message);
   }
```

Summary

There are many aspects to creating a personalized solution with SharePoint. You have covered many different areas from user profiles to audiences. You have looked at how to make sites more personalized by using the features of My Site. These tasks could be done with the user interface, but the new object model functionality provides the ability to perform all of these actions with code. In this chapter, you have seen how to:

- ❑ Manage users and user profile data
- ❑ Create Web Parts and web pages that use the built-in controls like the people picker

- View and query the users' change logs
- Create a user's My Site programmatically
- Create colleagues and memberships using the object model
- Update a site's audience data
- Create new rules for audiences in a site

Now that you have seen how to update many aspects of the personalization through code, the next chapter looks at how to perform enterprise searches. This personalization data can help return the correct search results and limit the visibility of what a user can view.