Using Active Directory in .NET

By **Paul D. Sheriff**

Sometimes your .NET applications need to interact with Microsoft Active Directory (AD) to authenticate users, get a list of users, retrieve groups, or determine which users are within which AD groups. There are a few different approaches you can use to retrieve information from your AD database within your domain.

One approach is to utilize the Lightweight Directory Access Protocol (LDAP) using the DirectoryEntry and DirectorySearch classes under the System.DirectoryServices namespace. Another approach is to use the complete set of class wrappers around AD under the System.DirectoryServices.AccountManagement namespace.

In this article, you will learn to use LDAP queries to retrieve information from your AD database. The LDAP classes are much faster and allow you to get at almost all of AD, whereas the wrapper classes only allow you to get at Users, Groups, and Computer objects in AD. You will find that the DirectoryEntry and DirectorySearcher objects are faster than the objects in the System.DirectoryServices.AccountManagement namespace.

A few definitions are in order before we get into the actual code. First off, AD is a database-based system that provides authentication, directory, policy, and other services in a Microsoft Windows environment. LDAP is a language for querying and modifying items within a directory service like AD database. It is important to note that LDAP is a standard language used to query any kind of directory service. AD is a Microsoft proprietary implementation of a directory service and, as such, has some custom extensions on top of the LDAP standard language.

Building the LDAP Connection String

The first thing you must do in order to connect to any directory service is to create an LDAP connection string. A connection string uses the following format:

```
LDAP://DC=|SERVER NAME|[,DC=|EXTENSION|]
```

The connection string for a domain named XYZ.NET looks like the following:

```
LDAP://DC=XYZ,DC=net
```

Instead of having to know your actual domain name, you can use the following generic code to query the LDAP server for the connection string.

```
private string GetCurrentDomainPath()
{
  DirectoryEntry de = new DirectoryEntry("LDAP://RootDSE");

  return "LDAP://" + de.Properties["defaultNamingContext"][0].ToString();
}
```

That code returns the following:

```
LDAP://DC=pdsa,DC=net
```

Get All Users

One of the first things you might wish to do is to retrieve all users from your AD. This is accomplished with a few different classes located within the System.DirectoryServices.dll and in the System.DirectoryServices namespace. The DirectoryEntry class is used to hold the LDAP connection string. The DirectorySearcher class is used to perform a search against the LDAP connection. You set the Filter property on the DirectorySearcher object to a valid LDAP query. Calling the FindAll() method on the DirectorySearcher object returns a SearchResultCollection object. This collection of SearchResult objects contains the values retrieved from the AD.

Listing 1 shows the complete code you need to retrieve all users from your AD domain. By default, the only property returned from your AD database is the "name" property. If you are familiar with AD, you know that users can be created with email address, first name, middle name, last name, and many other properties. There are additional steps that you must perform if you wish to retrieve these properties that will be discussed in the next section.
What is interesting in **Listing 1** is that you reference the "name" property using [0]. You would think that there can only be one name, and you are correct. However, because SearchResult is a generic object that could contain any type of AD object, each of the properties could have more than one value. For instance, if you retrieve a Group object from AD, one of the properties "members" contains an array of member names that make up that group. Each property you retrieve needs to use the index of 0, or if that property is a group, you can loop through that property's array by incrementing the index number until you reach the end of the array.

Retrieve Additional User Info

To keep things lightweight, LDAP only retrieves the most basic of information, such as the name of a User, Group, Organization Unit, etc. You may request more information, but you must tell LDAP what you wish to retrieve prior to performing your search. In the DirectorySearcher object this is accomplished by adding property names to the PropertiesToLoad property. You must know the names of the properties that are available in your directory service. **Listing 2** shows adding the appropriate properties for Microsoft's AD database to retrieve information such as the user's first name (givenname), last name (sn), email address (mail), and login name (userPrincipalName). An additional property you might find useful is distinguishedName. This gives you the full LDAP query for that particular user. This LDAP query looks similar to this.

```
CN=Person,CN=Bruce Jones,DC=XZY,DC=net
```

Notice the "if" statement in the code in **Listing 2** prior to displaying any of the properties. The reason for this is that these properties are optional within AD and if you don't perform the check, you could get a null reference exception. You don't need to put an "if" statement around the name and distinguishedName properties because these will always be there, but you might want to keep things consistent.

Build a UserSearcher Method

As you can imagine, you will probably create a DirectorySearcher for retrieving users in many places. It's a good idea to create a method that creates the DirectorySearcher object for you and populates it with the list of properties that you are interested in. **Listing 3** shows a method called BuildUserSearcher to which you will pass in a DirectoryEntry object. An instance of a DirectorySearcher object is created, and then the properties are added to the PropertiesToLoad property.

Build Extension Method for Reading Properties

Another tedious task in the code shown in **Listing 2** is constantly having to check for the existence of a property prior to retrieving it. Instead of writing this code over and over, create an extension method that either returns the property value or an empty string if the property is not found. The next code snippet is an extension method for the SearchResult class.

```
public static class ADExtensionMethods
{
  public static string GetPropertyValue(this SearchResult sr, string propertyName)
  {
    string ret = string.Empty;

    if (sr.Properties[propertyName].Count > 0)
        ret = sr.Properties[propertyName][0].ToString();

    return ret;
  }
}
```

The extension method can replace the code with all of the "if" statements to look like the following:

```
foreach (SearchResult sr in results)
{
  Debug.WriteLine(sr.GetPropertyValue("name"));
  Debug.WriteLine(sr.GetPropertyValue("mail"));
  Debug.WriteLine(sr.GetPropertyValue("givenname"));
  Debug.WriteLine(sr.GetPropertyValue("sn"));
  Debug.WriteLine(sr.GetPropertyValue("userPrincipalName"));
  Debug.WriteLine(sr.GetPropertyValue("distinguishedName"));
}
```

Searching for Users

Instead of getting all users, you might wish to retrieve just a subset of users. This can be accomplished quite easily. as shown in **Listing 4**. You only need to add one additional LDAP query to the Filter property. Adding (name=P*) searches for all users with a name that begins with the letter P.

```
ds.Filter =
      "(&(objectCategory=User)(objectClass=person)
       (name=" + userName + "*))";
```

In **Listing 4,** you can see an example method to which you will pass a complete or partial user name. You build a DirectorySearcher object and set the filter as described in the code snippet above. The rest of the code is the same as presented in **Listing 2** except you are now using the BuildUserSearcher method and the extension method to retrieve a property of the user.

Get One User

You can search for a specific user by using the previous technique of adding an LDAP query. Just eliminate the asterisk (*) from the query in order to do an exact match. Notice that the code in **Listing 5** uses a SearchResult instead of a SearchResultCollection. Call the FindOne method instead of FindAll because you are interested in retrieving a single user and not a list. If the user is not found, a null SearchResult object is returned.

Get All Groups

As you have seen in all of the previous code, you are always using the same set of classes for all your querying needs. Namely; DirectoryEntry, DirectorySearcher, SearchResult, and SearchResultCollection. These classes handle almost all of your querying needs. Look at the code in **Listing 6** to see an example of retrieving all Groups from Active Directory. You will notice that the code is almost identical, except the Filter property has a different LDAP query. I added another new option to this code and that is the ability to set a Sort option. You will most likely wish to retrieve your results in some sorted order. You can specify any property to sort the data by creating a SortOption object passing in the name of the property to sort upon.
The code in **Listing 6** shows one of the features of the SearchResult property that I discussed earlier: the ability for a property to contain an array of additional information. A group can be a member of another group and a group can contain members. Specify whether you wish to retrieve these additional properties by adding them to the PropertiesToLoad property just like you did for retrieving additional user properties. After performing the FindAll method, all the data is retrieved and all you have to do is to check to see if there is data within the memberof and member arrays. Loop though the data and display all of the groups of which this group is a member and the members of this group.

Creating a Login Screen

Another possible use of these AD objects you have been learning is to authenticate a user against an AD. To accomplish this, you first build a login screen such as the one shown in **Figure 1**. Next you write code to validate that the domain, user name, and password are valid credentials within the Active Directory.

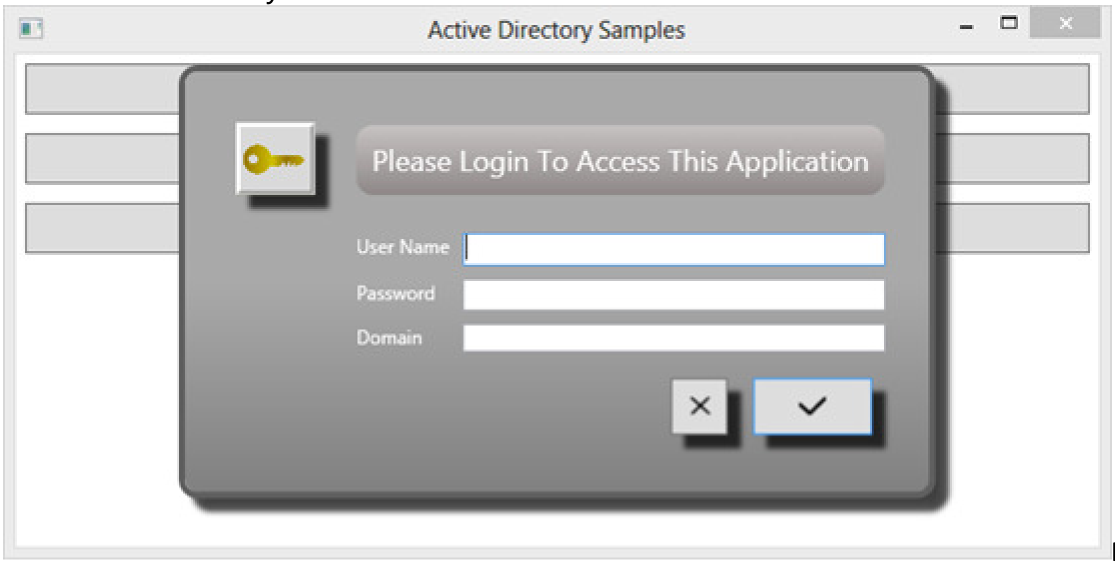

Figure 1: Create a login screen in WPF and use the AD objects to authenticate.

The AuthenticateUser Method

In order to authenticate a user against your Active Directory, you supply a valid LDAP path string to the constructor of the DirectoryEntry class. The LDAP path string is in the format LDAP://DomainName. You also pass in the user name and password to the constructor of the DirectoryEntry class. Pass this DirectoryEntry object to the constructor of a DirectorySearcher object and call the FindOne method. If the DirectorySearcher object returns a SearchResult, the credentials supplied are valid. If the credentials are not valid on the Active Directory, an exception is thrown. The code to authenticate is shown in **Listing 7**.
In the login window creates the following code under the Click event procedure of the Login button.

```
private void btnLogin_Click(object sender,RoutedEventArgs e)
{
  if (AuthenticateUser(txtDomain.Text, txtUserName.Text,
                           txtPassword.Password))
    DialogResult = true;
  else
    MessageBox.Show("Unable to Authenticate Using the Supplied Credentials");
}
```

Displaying the Login Screen

When you wish to authenticate a user, display the login screen shown in **Figure 1** modally within your application. The snippet below is the code to display the login window (named winLogin in the sample application).

```
private void DisplayLoginScreen()
{
  winLogin win = new winLogin();

  win.Owner = this;
  win.ShowDialog();
  if (win.DialogResult.HasValue && win.DialogResult.Value)
    MessageBox.Show("User Logged In");
  else
    MessageBox.Show("User NOT Logged In");
}
```

Because this code is called from another window within your WPF application, you set the owner of the login screen to the current window. Call the ShowDialog method on the login screen to have the login form displayed modally. After the user clicks on one of the two buttons, you need to check to see what the DialogResult property was set to. The DialogResult property is a nullable type, and thus you first need to check to see if the value has been set. After retrieving the result, you can now perform whatever code is appropriate for your application.

Summary

In this article, you learned how to query Active Directory to retrieve users, groups and even to authenticate a user. With just a few classes and some basic LDAP queries you can quickly retrieve information from your AD database. There is much more you can do with LDAP queries, such as adding, editing, and deleting information in your AD. I'll leave that up to you to explore these additional topics. You can find many blog posts and articles on the Web that discuss how to perform these actions on AD.

NOTE: You can download the complete sample code at my website
http://www.pdsa.com/downloads
and choose "PDSA Articles", then "Code Magazine - Using Active Directory from .NET" from the drop-down list.

Using the System.DirectoryServices.AccountManagement Classes

Microsoft realizes that not everyone wants to learn the LDAP query language. They created a set of objects that automatically populates collections of Users, Groups, and Computers in a set of strongly-typed classes. Instead of you having to type some fairly persnickety queries, you can use some fairly well laid-out objects. An example of retrieving all users utilizing these management classes looks like the following code snippet:

```
PrincipalContext domainContext = new PrincipalContext(ContextType.Domain);

UserPrincipal up = new UserPrincipal(domainContext);

PrincipalSearcher searcher = new PrincipalSearcher(up);

PrincipalSearchResult<Principal> result =

searcher.FindAll();

foreach (Principal item in result)

{

        Debug.WriteLine(item.Name);

}
```

A UserPrincipal object contains properties such as Name, EmailAddress, EmployeeId, GivenName, SurName, etc. Instead of you having to remember the strings to type in, such as "sn", "givenname", etc., these management objects have real properties. This is nice because if you misspell these property names, you get a compile-time error. Using the LDAP approach, you could get a runtime error, or at the very least, no data. There are GroupPrincipal and ComputerPrincipal objects that have similar properties for returning information about these types of objects within AD as well.

While having these objects does make accessing AD easier from a programming standpoint, it does come at a cost. To get additional properties beyond what is already on these classes means quite a bit of extra programming. In addition, you will find the management classes to be much slower compared to the LDAP equivalents.

Listing 1: With just a few lines of code, you can retrieve all users within your Active Directory.

```
private void GetAllUsers()
{
  DirectoryEntry de = new DirectoryEntry(GetCurrentDomainPath());
  DirectorySearcher ds = new DirectorySearcher(de);

  ds.Filter = "(&(objectCategory=User)(objectClass=person))";

  SearchResultCollection results = ds.FindAll();

  foreach (SearchResult sr in results)
  {
      // Using the index zero (0) is required!
      Debug.WriteLine(sr.Properties["name"][0].ToString());
  }
}
```

Listing 2: Add property names to the PropertiesToLoad property to retrieve those values from your directory service.

```
private void GetAdditionalUserInfo()
{
  DirectoryEntry de = new DirectoryEntry(GetCurrentDomainPath());
  DirectorySearcher ds = new DirectorySearcher(de);

  // Full Name
  ds.PropertiesToLoad.Add("name");
  // Email Address
  ds.PropertiesToLoad.Add("mail");
  // First Name
  ds.PropertiesToLoad.Add("givenname");
  // Last Name (Surname)
  ds.PropertiesToLoad.Add("sn");
  // Login Name
  ds.PropertiesToLoad.Add("userPrincipalName");
  // Distinguished Name
  ds.PropertiesToLoad.Add("distinguishedName");

  ds.Filter = "(&(objectCategory=User)(objectClass=person))";

  SearchResultCollection results = ds.FindAll();

  foreach (SearchResult sr in results)
  {
    if (sr.Properties["name"].Count > 0)
      Debug.WriteLine(sr.Properties["name"][0].ToString());
    // If not filled in, then you will get an error
    if (sr.Properties["mail"].Count > 0)
      Debug.WriteLine(sr.Properties["mail"][0].ToString());
    if (sr.Properties["givenname"].Count > 0)
      Debug.WriteLine(
            sr.Properties["givenname"][0].ToString());
    if (sr.Properties["sn"].Count > 0)
      Debug.WriteLine(sr.Properties["sn"][0].ToString());
    if (sr.Properties["userPrincipalName"].Count > 0)
      Debug.WriteLine(
            sr.Properties["userPrincipalName"][0].ToString());
    if (sr.Properties["distinguishedName"].Count > 0)
      Debug.WriteLine(
            sr.Properties["distinguishedName"][0].ToString());
  }
}
```

Listing 3: Create a generic method that you can call over and over to load those properties you wish to retrieve for an AD user.

```
private DirectorySearcher BuildUserSearcher(DirectoryEntry de)
{
  DirectorySearcher ds = new DirectorySearcher(de);

  // Full Name
  ds.PropertiesToLoad.Add("name");
  // Email Address
  ds.PropertiesToLoad.Add("mail");
  // First Name
  ds.PropertiesToLoad.Add("givenname");
  // Last Name (Surname)
  ds.PropertiesToLoad.Add("sn");
  // Login Name
  ds.PropertiesToLoad.Add("userPrincipalName");
  // Distinguished Name
  ds.PropertiesToLoad.Add("distinguishedName");

  return ds;
}
```

Listing 4: Add an additional LDAP query to the Filter property to perform a fuzzy search for users.

```
private void SearchForUsers(string userName)
{
  DirectoryEntry de = new DirectoryEntry(GetCurrentDomainPath());
  DirectorySearcher ds = BuildUserSearcher(de);

  ds.Filter = "(&(objectCategory=User)(objectClass=person)
               (name=" + userName + "*))";

  SearchResultCollection results = ds.FindAll();

  foreach (SearchResult sr in results)
  {
    Debug.WriteLine(sr.GetPropertyValue("name"));
    Debug.WriteLine(sr.GetPropertyValue("mail"));
    Debug.WriteLine(sr.GetPropertyValue("givenname"));
    Debug.WriteLine(sr.GetPropertyValue("sn"));
    Debug.WriteLine(sr.GetPropertyValue("userPrincipalName"));
    Debug.WriteLine(sr.GetPropertyValue("distinguishedName"));
  }
}
```

Listing 5: Pass in a single user name and call the FindOne method to retrieve a single user from AD.

```
private void GetAUser(string userName)
{
  DirectoryEntry de = new DirectoryEntry(GetCurrentDomainPath());
  DirectorySearcher ds = BuildUserSearcher(de);

  ds = BuildUserSearcher(de);
  ds.Filter = "(&(objectCategory=User)(objectClass=person)
               (name=" + userName + ")";

  SearchResult sr = ds.FindOne();
  if (sr != null)
  {
    Debug.WriteLine(sr.GetPropertyValue("name"));
    Debug.WriteLine(sr.GetPropertyValue("mail"));
    Debug.WriteLine(sr.GetPropertyValue("givenname"));
    Debug.WriteLine(sr.GetPropertyValue("sn"));
    Debug.WriteLine(sr.GetPropertyValue(
                      "userPrincipalName"));
    Debug.WriteLine(sr.GetPropertyValue(
                      "distinguishedName"));
  }
}
```

Listing 6: Retrieving AD Groups is very similar to retrieving users. The difference is in the LDAP query you supply to the Filter property.

```
private void GetAllGroups()
{
  DirectoryEntry de = new DirectoryEntry(GetCurrentDomainPath());
  DirectorySearcher ds = new DirectorySearcher(de);

  // Sort by name
  ds.Sort = new SortOption("name", SortDirection.Ascending);
  ds.PropertiesToLoad.Add("name");
  ds.PropertiesToLoad.Add("memberof");
  ds.PropertiesToLoad.Add("member");

  ds.Filter = "(&(objectCategory=Group))";

  SearchResultCollection results = ds.FindAll();

  foreach (SearchResult sr in results)
  {
    if (sr.Properties["name"].Count > 0)
      Debug.WriteLine(sr.Properties["name"][0].ToString());

    if (sr.Properties["memberof"].Count > 0)
    {
      Debug.WriteLine(" Member of...");
      foreach (string item in sr.Properties["memberof"])
      {
        Debug.WriteLine(" " + item);
      }
    }
    if (sr.Properties["member"].Count > 0)
    {
      Debug.WriteLine(" Members");
      foreach (string item in sr.Properties["member"])
```

```
            {
              Debug.WriteLine(" " + item);
            }
        }
    }
}
```

Listing 7: Authenticating a user is as simple as using an overload of the DirectoryEntry constructor and passing in the Domain, the user name, and the password.

```
private bool AuthenticateUser(string domainName, string userName, string password)
{
  bool ret = false;

  try
  {
    DirectoryEntry de = new DirectoryEntry("LDAP://" + domainName,
                                           userName, password);
    DirectorySearcher ds = new DirectorySearcher(de);
    SearchResult sr = ds.FindOne();

    ret = true;
  }
  catch
  {
    ret = false;
  }

  return ret;
}
```