Active Directory With C#

Introduction

If you work in the kind of large institution that I do and are using Microsoft Active Directory then the chances are that at certain times you will need to perform actions on the directory that are outside the scope of the MSAD tools. This could be things like specialised queries, bulk account creation or mass updates of user information. The MSAD tools and even some of the command line tools are quite limiting and difficult to use in this regard.

Whatever the reason, you may find that at some point you need to either purchase additional software for managing AD or write your own. Obviously I'd rather write my own software as it's cheaper, more rewarding and you can customise it however you like!

I found that when I was trying to learn how to make C# work nicely with AD there were a lack of simple tutorials to get me started, although I did find a few useful blog posts. Often any examples that I found did much more in the program than I was after, so it was difficult to pick out the few lines that I was actually interested in.

So, this page contains a few basic but fully working programs which illustrate common scenarios that you may have. If you can read and understand these examples you should be able to apply the principles to much larger and very powerful programs as I have done.

Obviously you need to be careful with this kind of programming and where ever possible you shouldn't be testing on a live environment. Queries are safe enough but when you get on to account creation and modification the potential to royally muck up a lot of account very quickly is a real danger, so take care!

Two Approaches

What is a little bit confusing is that there are essentially two sets of classes which can be used for AD operations. One is easier to use but not as versatile, the other is harder to use but lets you do pretty much anything (within my experience anyway!).

Which approach you use will depend on your project requirements. If you literally want to write a password resetting tool or a simple phone book then the `AccountManagement` libraries probably contain everything you need so you should use those.

For anything more complicated you may find that you need to get a bit more down and dirty with the LDAP and use the `DirectoryServices` approach.

System.DirectoryServices

MSDN Documentation

The 'older' and more difficult approach is using just `System.DirectoryServices` on its own. This lets you do pretty much anything that you like however the approach is more technical.

For example the properties of the AD objects (description, telephone etc.) are all held in an array which can present its own problems and involve a lot of iteration and use of casting since they are all generic objects.

Using this approach doing things like setting the password or enabling/disbaling the account is much more cryptic in the way in which it is achieved, often requiring UAC codes to be manually set and so on.

System.DirectoryServices.AccountManagement

MSDN Documentation

The 'newer' approach is to use `System.DirectoryServices.AccountManagement` which was designed to make managing AD through .NET much easier. Rather than accessing properties using an array they are exposed directly within the classes (and typed accordingly), allowing us to use things like `user.DisplayName` which is much tidier.

We also have easy to use methods available such as `.SetPassword()` and `.UnlockAccount()` as well as the `.Enabled` property which can be used to easily manage accounts. These are self explanatory in use once you have retrieved the object from AD so are not included in the examples!

The problem as I said though is that whilst the `AccountManagement` library makes things much easier in some regards it is also quite limited in others.

It exposes only a small number of the LDAP fields that you may want to use (name, description, email, home dir and phone is about it) so if you need access to a more obscure property it won't suffice.

Mixed Approach

One thing that's worth noting is that you can use the newer libraries to get a `UserPrincipal` object as they are called, and then access the underlying LDAP object with the `.GetUnderlyingObject()` method. This means that you could start a program with the newer approach but if you find it too limiting drop into the older approach half way through and have full access.

This is not a very neat approach but does work well, we will have a look at how this works in example 8. Hopefully if the newer libraries are expanded further in future .NET released there should be less and less reason to ever need to do this!

System.DirectoryServices Examples

These first examples all use the older approach and will serve you best if you are writing a large or complex AD management program.

Adjusting the Filter

In all of the examples where the program asks for a username the program then matches this to the field `cn`, which is what the AD GUI refers to as 'Full Name' and is what is listed as 'name' in the tabulated account lising of Active Directory Users and Computers.

You could change the username to something else by adjusting the filter. For example if you wanted to enter a user logon name (called `samaccountname` in the schema), you could set the filter as follows:

```
search.Filter = "(samaccountname=" + username + ")";
```

The createDirectoryEntry Function

All of these examples contain the same function called `createDirectoryEntry`, located at the bottom of the program. In order to try out the examples you will need to edit this function and enter both a hostname for your own AD server and also an appropriate search path. I have left in as examples the paths that I used when creating the programs.

If you are logged into a system as a domain administrator or a user with appropriate privilages then you should not need to specify a username and password for the connection.

However, if you are running the program as an unprivilaged user then you will need to add (or prompt for and program accordingly) a username and password to the `DirectoryEntry` object. The function is overloaded several times so you can just append as follows:

```
DirectoryEntry ldapConnection = new DirectoryEntry("server", "username",
"password");
```

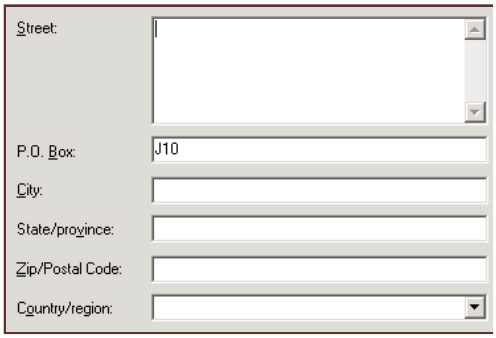
Example one : Retrieving All Information From a User's Record

This first example will introduce you to the classes needed for querying the AD using C#. I will explain this example fully as this will give a good understanding of the other examples also, once you grasp the major principles involved.

What we are going to do first is retrieve a full LDAP entry for a particular user. This isn't something that you would want to do very often as it isn't at all selective and would be overkill when querying a lot of users.

This is useful however if you need to find out what a particular field in the Active Directory is called.

For example, in the AD GUI we can set a 'PO Box' as part of the address (in College we use this for pigeon hole numbers). When you wish to query this information in your C# program the field is actually called `postofficebox`.



There is no tool that I know of which shows the correlation between the fields in the GUI and what the fields are called in the schema, so it has been necessary for me several times during development to set one of the fields to 'foo' and then run a full query looking for 'foo' in order to reveal the correct field.

The example is not too hard to understand, however there are several different classes used in order to accomplish the task. First we create a `DirectoryEntry` object. As you will have guessed from the section above regarding your setting, this class will contain all of the information which describes the server we are trying to connect to such as address, username and so on.

We then create a `DirectorySearcher` object. This class describes a search and operates against the `DirectoryEntry` object, so it knows *where* to search, and has it's own properties such as its `Filter` so it knows *what* to search for.

We then use the class `SearchResult` against the `DirectorySearcher` object, which represents an LDAP entry. This object has a number of `Properties` (such as user name, e-mail address) and a number of generic objects associated with each property:

SearchResult result	
Properties	**Objects**
cn	Ian Atkinson
mail	santa@clause.ac.uk
memberof	users
	staff
	domain administrators
...	...

The properties have generic objects associated with them as the class has no concept of their content. If you wish you will need to cast or convert to more specific classes in order to perform some operations, for example a telephone extension could be cast to an `int`.

In many cases there will be a single object associated with each property, for example a user can have only one user logon name (or `samaccountname`).

However some properties, such as `memberof` which represents a user's group membership, will have many objects (one for each group in this case).

The `SearchResult` object operates like an array, so we can retrieve a particular value such as `result.Properties["cn"][0]` for the first object associated with the `cn` property. In the example above `result.Properties["memberof"][1]` is "staff".

We can also iterate through all of the objects associated with a given property by using the `ResultPropertyCollection` class, which is what we do in the example below.

NB: this first example is more heavily commented than the rest in order to outline the common parts. In subsequent examples I have removed the comments and only commented the new or relevant parts.

retrieve_all_info.cs:

view plain

```
01. using System;
02. using System.Text;
03. using System.DirectoryServices;
04.
05. namespace activeDirectoryLdapExamples
06. {
07.    class Program
08.    {
09.       static void Main(string[] args)
10.       {
11.          Console.Write("Enter user: ");
12.          String username = Console.ReadLine();
13.
14.          try
15.          {
16.             // create LDAP connection object
17.
18.             DirectoryEntry myLdapConnection = createDirectoryEntry();
19.
20.             // create search object which operates on LDAP connection object
21.             // and set search object to only find the user specified
22.
23.             DirectorySearcher search = new DirectorySearcher(myLdapConnection);
24.             search.Filter = "(cn=" + username + ")";
25.
```

```
            // create results objects from search object

            SearchResult result = search.FindOne();

            if (result != null)
            {
               // user exists, cycle through LDAP fields (cn, telephonenumber etc.)

               ResultPropertyCollection fields = result.Properties;

               foreach (String ldapField in fields.PropertyNames)
               {
                  // cycle through objects in each field e.g. group membership
                  // (for many fields there will only be one object such as name)

                  foreach (Object myCollection in fields[ldapField])
                     Console.WriteLine(String.Format("{0,-20} : {1}",
                                 ldapField, myCollection.ToString()));
               }
            }

            else
            {
               // user does not exist
               Console.WriteLine("User not found!");
            }
         }

         catch (Exception e)
         {
            Console.WriteLine("Exception caught:\n\n" + e.ToString());
         }
      }

      static DirectoryEntry createDirectoryEntry()
      {
         // create and return new LDAP connection with desired settings

         DirectoryEntry ldapConnection     = new DirectoryEntry("rizzo.leeds-art.ac.uk");
         ldapConnection.Path               = "LDAP://OU=staffusers,DC=leeds-art,DC=ac,DC=uk";
         ldapConnection.AuthenticationType = AuthenticationTypes.Secure;

         return ldapConnection;
      }
   }
}
```

Here is an (abbreviated) example of the output:

```
H:\Desktop\adcsharp>retrieve_all_info

Enter user: Ian Atkinson

distinguishedname    : CN=Ian Atkinson,OU=IT,OU=staffusers,DC=leeds-art,DC=ac,D
cn                   : Ian Atkinson
mailnickname         : iana
displayname          : Ian Atkinson
title                : Senior Infrastructure Support Engineer
samaccountname       : iana
givenname            : Ian
mail                 : santa@clause.ac.uk
sn                   : Atkinson
postofficebox        : J10

<snip>
```

Example 2 - Retrieving Selected Information From a User's Record

This example is almost identical to the above example, however we are now selective about which fields from the AD we want to bring in. This is a much more realistic example as it's obviously bad practise to query more data than is required.

We load certain properties by calling the `PropertiesToLoad.Add` method on our `DirectorySearcher` object.

retrieve_some_info.cs:

view plain

```
using System;
using System.Text;
using System.DirectoryServices;

namespace activeDirectoryLdapExamples
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Enter user: ");
         String username = Console.ReadLine();

         try
         {
            DirectoryEntry myLdapConnection = createDirectoryEntry();
            DirectorySearcher search = new DirectorySearcher(myLdapConnection);
            search.Filter = "(cn=" + username + ")";

            // create an array of properties that we would like and
            // add them to the search object

            string[] requiredProperties = new string[]{"cn", "postofficebox", "mail"};

            foreach (String property in requiredProperties)
               search.PropertiesToLoad.Add(property);

            SearchResult result = search.FindOne();

```

```
            if (result != null)
            {
               foreach (String property in requiredProperties)
                  foreach (Object myCollection in result.Properties[property])
                     Console.WriteLine(String.Format("{0,-20} : {1}",
                                 property, myCollection.ToString()));
            }

            else Console.WriteLine("User not found!");
         }

         catch (Exception e)
         {
            Console.WriteLine("Exception caught:\n\n" + e.ToString());
         }
      }

      static DirectoryEntry createDirectoryEntry()
      {
         // create and return new LDAP connection with desired settings

         DirectoryEntry ldapConnection     = new DirectoryEntry("rizzo.leeds-art.ac.uk");
         ldapConnection.Path               = "LDAP://OU=staffusers,DC=leeds-art,DC=ac,DC=uk";
         ldapConnection.AuthenticationType = AuthenticationTypes.Secure;
         return ldapConnection;
      }
   }
}
```

Here is an example of the output:

```
H:\Desktop\adcsharp>retrieve_some_info

Enter user           : Ian Atkinson
cn                   : Ian Atkinson
postofficebox        : J10
mail                 : santa@clause.ac.uk
```

Example 3 - Retrieving Information for All Users

So far we have only retrieved information for a single user. In this example we will retrieve some information for all of the users in our search base.

We can accomplish this simply by using the `FindAll` rather than the `FindOne` method on our `DirectorySearcher` object and then iterating through the results.

all_users.cs:

view plain

```
using System;
using System.Text;
using System.DirectoryServices;

namespace activeDirectoryLdapExamples
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Enter property: ");
         String property = Console.ReadLine();

         try
         {
            DirectoryEntry myLdapConnection = createDirectoryEntry();

            DirectorySearcher search = new DirectorySearcher(myLdapConnection);
            search.PropertiesToLoad.Add("cn");
            search.PropertiesToLoad.Add(property);

            SearchResultCollection allUsers = search.FindAll();

            foreach(SearchResult result in allUsers)
            {
               if (result.Properties["cn"].Count > 0 && result.Properties[property].Count > 0)
               {
                  Console.WriteLine(String.Format("{0,-20} : {1}",
                              result.Properties["cn"][0].ToString(),
                              result.Properties[property][0].ToString()));
               }
            }
         }

         catch (Exception e)
         {
            Console.WriteLine("Exception caught:\n\n" + e.ToString());
         }
      }

      static DirectoryEntry createDirectoryEntry()
      {
         // create and return new LDAP connection with desired settings

         DirectoryEntry ldapConnection = new DirectoryEntry("rizzo.leeds-art.ac.uk");
         ldapConnection.Path = "LDAP://OU=staffusers,DC=leeds-art,DC=ac,DC=uk";
         ldapConnection.AuthenticationType = AuthenticationTypes.Secure;
         return ldapConnection;
      }
   }
}
```

Here is an example of the output:

```
H:\Desktop\adcsharp>all_users

Enter property       : mail

Ian Atkinson         : santa@clause.ac.uk
Rudolph              : rudolph@clause.ac.uk
Elf                  : elf@clause.ac.uk
```

Example 4 - Updating a User

Having covered querying the AD we will now move on to updating the AD! This is much simpler than you might imagine as the search results that we have already found really represent actual objects on the server, so we can easily edit the properties of the result and then write this information back to the AD.

We do this by creating a `DirectoryEntry` object from the search result (using the `GetDirectoryEntry` method) and then setting the `Value` for any property that we would like to change. When we are finished we use the `CommitChanges` method to actually write the changes.

In this small example we retrieve a user's job title (`title` in the schema) and then change it for a new one.

update_user.cs:

view plain

```
01. using System;
02. using System.Text;
03. using System.DirectoryServices;
04.
05. namespace activeDirectoryLdapExamples
06. {
07.     class Program
08.     {
09.         static void Main(string[] args)
10.         {
11.             Console.Write("Enter user      : ");
12.             String username = Console.ReadLine();
13.
14.             try
15.             {
16.                 DirectoryEntry myLdapConnection = createDirectoryEntry();
17.
18.                 DirectorySearcher search = new DirectorySearcher(myLdapConnection);
19.                 search.Filter = "(cn=" + username + ")";
20.                 search.PropertiesToLoad.Add("title");
21.
22.                 SearchResult result = search.FindOne();
23.
24.                 if (result != null)
25.                 {
26.                     // create new object from search result
27.
28.                     DirectoryEntry entryToUpdate = result.GetDirectoryEntry();
29.
30.                     // show existing title
31.
32.                     Console.WriteLine("Current title   : " +
33.                                       entryToUpdate.Properties["title"][0].ToString());
34.
35.                     Console.Write("\n\nEnter new title : ");
36.
37.                     // get new title and write to AD
38.
39.                     String newTitle = Console.ReadLine();
40.
41.                     entryToUpdate.Properties["title"].Value = newTitle;
42.                     entryToUpdate.CommitChanges();
43.
44.                     Console.WriteLine("\n\n...new title saved");
45.                 }
46.
47.                 else Console.WriteLine("User not found!");
48.             }
49.
50.             catch (Exception e)
51.             {
52.                 Console.WriteLine("Exception caught:\n\n" + e.ToString());
53.             }
54.         }
55.
56.         static DirectoryEntry createDirectoryEntry()
57.         {
58.             // create and return new LDAP connection with desired settings
59.
60.             DirectoryEntry ldapConnection     = new DirectoryEntry("rizzo.leeds-art.ac.uk");
61.             ldapConnection.Path               = "LDAP://OU=staffusers,DC=leeds-art,DC=ac,DC=uk'
62.             ldapConnection.AuthenticationType = AuthenticationTypes.Secure;
63.             return ldapConnection;
64.         }
65.     }
66. }
```

Here is an (abbreviated) example of the output. Note how when the program is run for the second time the title that is retrieved is the one entered the first time around:

```
H:\Desktop\adcsharp>update_user

Enter user      : Ian Atkinson
Current title   : Senior Infrastructure Support Engineer
```

```
Enter new title : Dogsbody

...new title saved

H:\Desktop\adcsharp>update_user

Enter user      : Ian Atkinson
Current title   : Dogsbody

Enter new title : Senior Infrastructure Support Engineer

...new title saved
```

Example 5 - Adding a New User

One of the most complex things that you may decide you need to do is add a new user from your C# program, rather than using the AD tools.

Again, there are various commercial programs to do this and also tools in the Resource Kit than can be scripted with, but you may find that you just can't find something to do absolutely everything that you need, just how you need it done.

The program below should be a good starting point for anyone wanting to add in their own users. It shows you how to:

- Create a user with custom options
- Set a password
- Enable the account
- Add the user to some groups
- Create a home folder
- Set the ownserhip of the home folder
- Set the permissions/ACL of the home folder

Obviously if you wanted to use this as a basis for your own program you would need to set the options to your own requirements and tweak as necessary.

Specifically if you want to write a flexible program to write users in and out of different OUs, rather than a single OU, then it will be necessary to create multiple LDAP connections with different paths, and also a more complex function to add users to groups which searches the whole subtree.

create_user.cs:

view plain

```
01. using System;
02. using System.Text;
03. using System.DirectoryServices;
04. using System.IO;
05. using System.Security.AccessControl;
06. using System.Security.Principal;
07.
08. namespace activeDirectoryLdapExamples
09. {
10.    class Program
11.    {
12.       static void Main(string[] args)
13.       {
14.          // connect to LDAP
15.
16.          DirectoryEntry myLdapConnection = createDirectoryEntry();
17.
18.          // define vars for user
19.
20.          String domain      = "leeds-art.ac.uk";
21.          String first       = "Test";
22.          String last        = "User";
23.          String description = ".NET Test";
24.          object[] password  = { "12345678" };
25.          String[] groups    = { "Staff" };
26.          String username    = first.ToLower() + last.Substring(0, 1).ToLower();
27.          String homeDrive   = "H:";
28.          String homeDir     = @"\\gonzo.leeds-art.ac.uk\data3\USERS\" + username;
29.
30.          // create user
31.
32.          try
33.          {
34.             if (createUser(myLdapConnection, domain, first, last, description,
35.                      password, groups, username, homeDrive, homeDir, true) == 0)
36.             {
37.
38.                Console.WriteLine("Account created!");
39.                Console.ReadLine();
40.             }
41.
42.             else
43.             {
44.                Console.WriteLine("Problem creating account :(");
45.                Console.ReadLine();
46.             }
47.          }
48.
49.          catch (Exception e)
50.          {
51.             Console.WriteLine("Exception caught:\n\n" + e.ToString());
52.             Console.ReadLine();
53.          }
54.       }
55.
56.       static int createUser(DirectoryEntry myLdapConnection, String domain, String first,
```

```
                            String last, String description, object[] password,
                            String[] groups, String username, String homeDrive,
                            String homeDir, bool enabled)
        {
            // create new user object and write into AD

            DirectoryEntry user = myLdapConnection.Children.Add(
                                "CN=" + first + " " + last, "user");

            // User name (domain based)
            user.Properties["userprincipalname"].Add(username + "@" + domain);

            // User name (older systems)
            user.Properties["samaccountname"].Add(username);

            // Surname
            user.Properties["sn"].Add(last);

            // Forename
            user.Properties["givenname"].Add(first);

            // Display name
            user.Properties["displayname"].Add(first + " " + last);

            // Description
            user.Properties["description"].Add(description);

            // E-mail
            user.Properties["mail"].Add(first + "." + last + "@" + domain);

            // Home dir (drive letter)
            user.Properties["homedirectory"].Add(homeDir);

            // Home dir (path)
            user.Properties["homedrive"].Add(homeDrive);

            user.CommitChanges();

            // set user's password

            user.Invoke("SetPassword", password);

            // enable account if requested (see http://support.microsoft.com/kb/305144 for other c

            if (enabled)
                user.Invoke("Put", new object[] { "userAccountControl", "512" });

            // add user to specified groups

            foreach (String thisGroup in groups)
            {
                DirectoryEntry newGroup = myLdapConnection.Parent.Children.Find(
                                        "CN=" + thisGroup, "group");

                if (newGroup != null)
                    newGroup.Invoke("Add", new object[] { user.Path.ToString() });
            }

            user.CommitChanges();

            // make home folder on server

            Directory.CreateDirectory(homeDir);

            // set permissions on folder, we loop this because if the program
            // tries to set the permissions straight away an exception will be
            // thrown as the brand new user does not seem to be available, it takes
            // a second or so for it to appear and it can then be used in ACLs
            // and set as the owner

            bool folderCreated = false;

            while (!folderCreated)
            {
                try
                {
                    // get current ACL

                    DirectoryInfo dInfo = new DirectoryInfo(homeDir);
                    DirectorySecurity dSecurity = dInfo.GetAccessControl();

                    // Add full control for the user and set owner to them

                    IdentityReference newUser = new NTAccount(domain + @"\" + username);

                    dSecurity.SetOwner(newUser);

                    FileSystemAccessRule permissions =
                        new FileSystemAccessRule(newUser, FileSystemRights.FullControl,
                                                 AccessControlType.Allow);

                    dSecurity.AddAccessRule(permissions);

                    // Set the new access settings.

                    dInfo.SetAccessControl(dSecurity);
                    folderCreated = true;
                }

                catch (System.Security.Principal.IdentityNotMappedException)
                {
                    Console.Write(".");
                }

                catch (Exception ex)
                {
                    // other exception caught so not problem with user delay as
                    // commented above

                    Console.WriteLine("Exception caught:" + ex.ToString());
                    return 1;
                }
```

```
169.          }
170.
171.          return 0;
172.       }
173.
174.       static DirectoryEntry createDirectoryEntry()
175.       {
176.          // create and return new LDAP connection with desired settings
177.
178.          DirectoryEntry ldapConnection     = new DirectoryEntry("rizzo.leeds-art.ac.uk");
179.          ldapConnection.Path               = "LDAP://OU=staffusers,DC=leeds-art,DC=ac,DC=uk";
180.          ldapConnection.AuthenticationType = AuthenticationTypes.Secure;
181.          return ldapConnection;
182.       }
183.    }
184. }
```

System.DirectoryServices.AccountManagement Examples

This second set of examples all use the newer libraries and will serve you best if you are writing smaller or simpler programs.

Example 6 - Retrieving Information From a User's Record

This first example is similar to example 2 but using the newer libraries. The approach to searching for a user is a little different as you can see. Here we will just retrieve a person's name and phone number from their logon name.

First a `PrincipalContext` object is created, this is the connection to AD and is overloaded many times so that you can pass it an LDAP path, user name and password if required (omitted from the example to keep it simple).

We then create a `UserPrincipal` object and set some criteria on it. This can be confusing as when we have retrieved a user from AD it will be an object of the same type, however this object is not real and is only used for searching. This effectively replaces the LDAP search filter. So we make the object and set its `samaccountname` which allows us to search in the `PrincipalContext` for a user with that logon name.

Finally we create the `PrincipalSearcher` object and run it using the search user we just created. The result is then cast into a new `UserPrincipal` object which represents the acutal AD user. The cast is used as the same approach is also used to find `GroupPrincipal` or `ComputerPrincipal` objects.

This may sound complicated but hopefully when you read below you will see it is only a few lines of code!

newer_details.cs:

view plain

```
01. using System;
02. using System.DirectoryServices.AccountManagement;
03.
04. namespace new_ad_exmaple
05. {
06.     class Program
07.     {
08.         static void Main(string[] args)
09.         {
10.             try
11.             {
12.                 // enter AD settings
13.                 PrincipalContext AD = new PrincipalContext(ContextType.Domain, "leeds-art.ac.uk
14.
15.                 // create search user and add criteria
16.                 Console.Write("Enter logon name: ");
17.                 UserPrincipal u = new UserPrincipal(AD);
18.                 u.SamAccountName = Console.ReadLine();
19.
20.                 // search for user
21.                 PrincipalSearcher search = new PrincipalSearcher(u);
22.                 UserPrincipal result = (UserPrincipal)search.FindOne();
23.                 search.Dispose();
24.
25.                 // show some details
26.                 Console.WriteLine("Display Name : " + result.DisplayName);
27.                 Console.WriteLine("Phone Number : " + result.VoiceTelephoneNumber);
28.             }
29.
30.             catch (Exception e)
31.             {
32.                 Console.WriteLine("Error: " + e.Message);
33.             }
34.         }
35.     }
36. }
```

And the output:

```
Enter logon name: iana
Display Name : Ian Atkinson
Phone Number : 1234
```

Example 7 - Retrieving Info From All Users Again

This example is similar to example 3, here we show all people's phone numbers. To find all objects we have still created the search object but have simply not specified any criteria for it (other than the AD it belongs to) so the search returns all objects. We can then iterate over these with a handy `foreach`.

In real life you should limit the scope of a search like this of course by specifying the correct OU rather than searching the whole AD, but I'm trying to keep the example code concise!

newer_phones.cs:

view plain

```
using System;
using System.DirectoryServices.AccountManagement;

namespace new_ad_exmaple
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                PrincipalContext  AD     = new PrincipalContext(ContextType.Domain, "leeds-art.
                UserPrincipal     u      = new UserPrincipal(AD);
                PrincipalSearcher search = new PrincipalSearcher(u);

                foreach(UserPrincipal result in search.FindAll())
                    if (result.VoiceTelephoneNumber != null)
                        Console.WriteLine("{0,30} {1} ", result.DisplayName, result.VoiceTelepl

                search.Dispose();
            }

            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
            }
        }
    }
}
```

And some abbreviated output:

```
                Joe Bloggs 1245
                Jim Bloggs 1456
                Jan Bloggs 1765
                Bob Bloggs 1298
```

Example 8 - Using Both Libraries

Here we will see how a `UserPrincipal` can be converted into a more general object so that we can access properties which are not exposed by the `AccountManagement` libraries. There should be no reason to do this in future if the libraries are expanded to expose more properties!

This example will now list the person's mail box along with the phone number by expanding the previous example, as I said earlier this is the field `postofficebox` in AD which is not exposed by `AccountManagement`.

In addition to the previous example we now get the underlying object from the `UserPrincipal` which we call `lowerLdap`.

Since `.GetUnderlyingObject()` returns a generic object we have to cast this to a `DirectoryEntry` (the `System.DirectoryServices` equivalent of a `UserPrincipal` as seen in the earlier examples above) so that we can access its `properties` array.

newer_dropdown.cs:

view plain

```
using System;
using System.DirectoryServices;
using System.DirectoryServices.AccountManagement;

namespace new_ad_exmaple
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                PrincipalContext  AD     = new PrincipalContext(ContextType.Domain, "leeds-art.
                UserPrincipal     u      = new UserPrincipal(AD);
                PrincipalSearcher search = new PrincipalSearcher(u);

                foreach (UserPrincipal result in search.FindAll())
                {
                    if (result.VoiceTelephoneNumber != null)
                    {
                        DirectoryEntry lowerLdap = (DirectoryEntry)result.GetUnderlyingObject()

                        Console.WriteLine("{0,30} {1} {2}",
                            result.DisplayName,
                            result.VoiceTelephoneNumber,
                            lowerLdap.Properties["postofficebox"][0].ToString());
                    }
                }

                search.Dispose();
            }

            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
            }
        }
    }
}
```

And some abbreviated output:

```
                  Joe Bloggs 1245 J10
                  Jim Bloggs 1456 J11
                  Jan Bloggs 1765 J12
                  Bob Bloggs 1298 J13
```


Validation HTML5 | CSS