Howto: (Almost) Everything In Active Directory via C#

thund3rstruck
22 May 2008 CPOL

A collection of the most common Active Directory Tasks in C#

Table of Contents

Introduction

When it comes to programmatically accessing Microsoft's Active Directory a lot of people seem to have quite a difficult time tying all the pieces together to accomplish exactly what they want to. There are so many technologies available for communicating with LDAP that many programmers end up with a mix between COM+ ADSI calls and .NET class calls mixed into their code. ADSI code is so difficult to understand and follow that the creator of that code usually owns it for the entirety of it's lifecycle since no one else wants to support it.

This article attempts to tie together the most commonly used elements involved in Active Directory Management in the simplest, most clean manner possible. I interact with Active Directory in nearly all of my applications (web & forms) and I have had to solve a lot of integration issues for many customers. When I was starting out with this technology I had a lot of growing pains so this is an attempt to help those programmers who may have a need to interact with the Directory but do not want to have to become experts in the issue. However, certain rudimentary knowledge and concepts are required in order to utilize the code. You must be familiar with such terms as: distinguishedName, ldap paths, fully qualified domain names, object attributes (single string & multi-string), and general knowledge of ldap schemas.

Background

There is a great collection of sample code available on MSDN's website for the v1.1 `System.DirectoryServices` assembly but there seems to be a void when it comes to the new functionality available in the v2.0 `System.DirectoryServices.ActiveDirectory` assembly. Since this article's original publishing, Generics have gained widespread acceptance and I encourage anyone borrowing from this resource to replace the archaic ArrayList collections with List<T> or appropriate generic collections.

Points of Concern

In order to communicate with Active Directory one must take into account network security, business rules, and technological constraints. If you're using Active Directory code from an ASP.NET page you must ensure that the code has the appropriate level of permission to access and interact with the directory. For development purposes or proof of concept you can enable impersonation at the ASP.NET level (in web.config) and the IIS level and if the IIS server and the directory domain controller reside on the same machine this will work. However, if these entities are not co-located on the same server (as they never are in production) you can wrap the code around an impersonation class (such as the Zeta Impersonator which will execute the Directory calls under the token of the impersonated user. **It's strongly recommended that you do not do this for security reasons unless absolutely necessary.**. The authorized method for granting the ASP.NET application permission to the directory is by way of either a privileged IIS Application Pool running under the identity of a service account or by way of a COM+ entity running under the identity of a service account.

If you plan on running this code from a desktop assembly then you're going to want to ensure that your machine is attached to a domain and can communicate with that domain. The impersonation is not necessary if the user running the code has sufficient privileges on the domain.

Running Code in Batch Processes

It is also important to note that if you plan on running this code from an ASP.NET page in batch, ASP.NET will time out on you if you try to run batch processes from it's primary thread. There are several things to consider in this scenario but be aware that for example, if you're creating x number of accounts through an ASP.NET application (or performing any batch operation in general) that you must plan to use queues, a back-end scheduler, or some other mechanism outside the scope of the page itself to prevent timing out during your processes. As with any ASPNET design, it's never a good idea to use ASPNET itself for anything but the "View" part of the solution. The best architecture would queue tasks into a SQL database or something to that effect and then a back-end windows service or similar application would pick up the tasking and perform the actual Directory operations.

This is typically how I engineer Active Directory management solutions for customers.

A Note on Method Parameters

You will notice that most of the methods require the same parameters. Rather than identify each time I will outline them now:

- `friendlyDomainName`: the non qualified domain name (contoso - NOT contoso.com)
- `ldapDomain`: the fully qualified domain such as contoso.com or `dc=contoso,dc=com`
- `objectPath`: the fully qualified path to the object: `CN=user`, `OU=USERS`, `DC=contoso`, `DC=com`(same as `objectDn`)
- `objectDn`: the `distinguishedName` of the object: `CN=group`, `OU=GROUPS`, `DC=contoso`, `DC=com`
- `userDn`: the `distinguishedName` of the user: `CN=user, OU=USERS, DC=contoso, DC=com`
- `groupDn`: the `distinguishedName` of the group: `CN=group,OU=GROUPS,DC=contoso,DC=com`

A Note on System.DirectoryServices.DirectoryEntry

You'll notice in all the samples that we're binding directly to the `directoryEntry` and not specifying a server or credentials. If you do not want to use an impersonation class you can send credentials directly into the `DirectoryEntry` constructor. The impersonation class is helpful for those times when you want to use a `static` method and don't want to go through the trouble of creating a `DirectoryContext` object to hold these details. Likewise you may want to target a specific domain controller.

Target Specific Domain Controllers or Credentials

Everywhere in the code that you see: `LDAP://` you can replace with `LDAP://MyDomainControllerNameOrIpAddress` as well as everywhere you see a `DirectoryEntry` class being constructed you can send in specific credentials as well. This is especially helpful if you need to work on an Active Directory for which your machine is not a member of it's forest or domain or you want to target a DC to make the changes to.

C#

```
//Rename an object and specify the domain controller and credentials directly


public static void Rename(string server,
    string userName, string password, string objectDn, string newName)
{
    DirectoryEntry child = new DirectoryEntry("LDAP://" + server + "/" +
        objectDn, userName, password);
    child.Rename("CN=" + newName);
}
```

Managing local accounts with DirectoryEntry

It is important to note that you can execute some of these methods against a local machine as opposed to an Active Directory if needed by simply replacing the `LDAP://` string with `WinNT://` as demonstrated below

C#

```
//create new local account

DirectoryEntry localMachine = new DirectoryEntry("WinNT://" +
    Environment.MachineName);
DirectoryEntry newUser = localMachine.Children.Add("localuser", "user");
newUser.Invoke("SetPassword", new object[] { "3l!teP@$$w0RDz" });
newUser.CommitChanges();
Console.WriteLine(newUser.Guid.ToString());
localMachine.Close();
newUser.Close();
```

Managing local groups with DirectoryEntry

A few configuration changes need to be made to the code but it's pretty straightforward. Below you can see an example of using `DirectoryEntry` to enumerate the members of the local "administrator" group.

C#

```
DirectoryEntry localMachine = new DirectoryEntry
    ("WinNT://" + Environment.MachineName + ",Computer");
DirectoryEntry admGroup = localMachine.Children.Find
    ("administrators", "group");
object members = admGroup.Invoke("members", null);

foreach (object groupMember in (IEnumerable)members)
{
    DirectoryEntry member = new DirectoryEntry(groupMember);
    Console.WriteLine(member.Name);
}
```

Managing IIS with DirectoryEntry

In addition to managing local & directory services accounts, the versatile `DirectoryEntry` object can manage other network providers as well, such as IIS. Below is an example of how you can use `DirectoryEntry` to provision a new virtual directory in IIS.

C#

```
//Create New Virtual Directory in IIS with DirectoryEntry()


string wwwroot = "c:\\Inetpub\\wwwroot";
string virtualDirectoryName = "myNewApp";
string sitepath = "IIS://localhost/W3SVC/1/ROOT";

DirectoryEntry vRoot = new DirectoryEntry(sitepath);
DirectoryWntry vDir = vRoot.Children.Add(virtualDirectoryName,
                            "IIsWebVirtualDir");
vDir.CommitChanges();

vDir.Properties["Path"].Value = wwwroot + "\\" + virtualDirectoryName;
vDir.Properties["DefaultDoc"].Value = "Default.aspx";
vDir.Properties["DirBrowseFlags"].Value = 2147483648;
vDir.Commitchanges();
vRoot.CommitChanges();
```

Active Directory Code

The code below is broken apart logically into usage categories. Again, this is not intended to be a complete library, just the code that I use on a daily basis.

Active Directory Management

C#

```
//These methods require these imports

//You must add a references in your project as well

using System.DirectoryServices;
using System.DirectoryServices.ActiveDirectory;
```

Translate the Friendly Domain Name to Fully Qualified Domain

C#

```
public static string FriendlyDomainToLdapDomain(string friendlyDomainName)
{
    string ldapPath = null;
    try
    {
        DirectoryContext objContext = new DirectoryContext(
            DirectoryContextType.Domain, friendlyDomainName);
        Domain objDomain = Domain.GetDomain(objContext);
        ldapPath = objDomain.Name;
    }
    catch (DirectoryServicesCOMException e)
    {
        ldapPath = e.Message.ToString();
    }
    return ldapPath;
}
```

Enumerate Domains in the Current Forest

C#

```
public static ArrayList EnumerateDomains()
{
    ArrayList alDomains = new ArrayList();
    Forest currentForest = Forest.GetCurrentForest();
    DomainCollection myDomains = currentForest.Domains;

    foreach (Domain objDomain in myDomains)
    {
        alDomains.Add(objDomain.Name);
    }
    return alDomains;
}
```

Enumerate Global Catalogs in the Current Forest

C#

```
public static ArrayList EnumerateDomains()
{
    ArrayList alGCs = new ArrayList();
    Forest currentForest = Forest.GetCurrentForest();
    foreach (GlobalCatalog gc in currentForest.GlobalCatalogs)
    {
        alGCs.Add(gc.Name);
    }
    return alGCs;
}
```

Enumerate Domain Controllers in a Domain

C#

```
public static ArrayList EnumerateDomainControllers()
{
    ArrayList alDcs = new ArrayList();
    Domain domain = Domain.GetCurrentDomain();
    foreach (DomainController dc in domain.DomainControllers)
    {
        alDcs.Add(dc.Name);
    }
    return alDcs;
}
```

Create a Trust Relationship

C#

```
public void CreateTrust(string sourceForestName, string targetForestName)
{
    Forest sourceForest = Forest.GetForest(new DirectoryContext(
        DirectoryContextType.Forest, sourceForestName));

    Forest targetForest = Forest.GetForest(new DirectoryContext(
        DirectoryContextType.Forest, targetForestName));

    // create an inbound forest trust

    sourceForest.CreateTrustRelationship(targetForest,
        TrustDirection.Outbound);
}
```

Delete a Trust Relationship

C#

```
public void DeleteTrust(string sourceForestName, string targetForestName)
{
    Forest sourceForest = Forest.GetForest(new DirectoryContext(
        DirectoryContextType.Forest, sourceForestName));

    Forest targetForest = Forest.GetForest(new DirectoryContext(
        DirectoryContextType.Forest, targetForestName));

    // delete forest trust

    sourceForest.DeleteTrustRelationship(targetForest);
}
```

Enumerate Objects in an OU

The parameter `OuDn` is the Organizational Unit `distinguishedName` such as `OU=Users,dc=myDomain,dc=com`

C#

```
public ArrayList EnumerateOU(string OuDn)
{
    ArrayList alObjects = new ArrayList();
    try
    {
        DirectoryEntry directoryObject = new DirectoryEntry("LDAP://" + OuDn);
        foreach (DirectoryEntry child in directoryObject.Children)
        {
            string childPath = child.Path.ToString();
            alObjects.Add(childPath.Remove(0,7));
                //remove the LDAP prefix from the path

            child.Close();
            child.Dispose();
        }
        directoryObject.Close();
        directoryObject.Dispose();
    }
    catch (DirectoryServicesCOMException e)
    {
        Console.WriteLine("An Error Occurred: " + e.Message.ToString());
    }
    return alObjects;
}
```

Enumerate Directory Entry Settings

One of the nice things about the 2.0 classes is the ability to get and set a configuration object for your `directoryEntry` objects.

C#

```
static void DirectoryEntryConfigurationSettings(string domainADsPath)
{
    // Bind to current domain

    DirectoryEntry entry = new DirectoryEntry(domainADsPath);
    DirectoryEntryConfiguration entryConfiguration = entry.Options;

    Console.WriteLine("Server: " + entryConfiguration.GetCurrentServerName());
    Console.WriteLine("Page Size: " + entryConfiguration.PageSize.ToString());
    Console.WriteLine("Password Encoding: " +
        entryConfiguration.PasswordEncoding.ToString());
    Console.WriteLine("Password Port: " +
        entryConfiguration.PasswordPort.ToString());
    Console.WriteLine("Referral: " + entryConfiguration.Referral.ToString());
    Console.WriteLine("Security Masks: " +
        entryConfiguration.SecurityMasks.ToString());
    Console.WriteLine("Is Mutually Authenticated: " +
        entryConfiguration.IsMutuallyAuthenticated().ToString());
    Console.WriteLine();
```

```
    Console.ReadLine();
}
```

Active Directory Objects

C#

```
//These methods require these imports

//You must add a references in your project as well

using System.DirectoryServices;
```

Check for the Existence of an Object

This method does not need you to know the `distinguishedName`, you can concat strings or even guess a location and it will still run (and return `false` if not found).

C#

```
public static bool Exists(string objectPath)
{
    bool found = false;
    if (DirectoryEntry.Exists("LDAP://" + objectPath))
    {
        found = true;
    }
    return found;
}
```

Move an Object from one Location to Another

It should be noted that the string `newLocation` should NOT include the `CN= value` of the object. The method will pull that from the `objectLocation` string for you. So object `CN=group,OU=GROUPS,DC=contoso,DC=com` is sent in as the `objectLocation` but the `newLocation` is something like: `OU=NewOUParent,DC=contoso,DC=com`. The method will take care of the `CN=group`.

C#

```
public void Move(string objectLocation, string newLocation)
{
    //For brevity, removed existence checks

    DirectoryEntry eLocation = new DirectoryEntry("LDAP://" + objectLocation);
    DirectoryEntry nLocation = new DirectoryEntry("LDAP://" + newLocation);
    string newName = eLocation.Name;
    eLocation.MoveTo(nLocation, newName);
    nLocation.Close();
```

```
    eLocation.Close();
}
```

Enumerate Multi-String Attribute Values of an Object

This method includes a recursive flag in case you want to recursively dig up properties of properties such as enumerating all the member values of a group and then getting each member group's groups all the way up the tree.

C#

```
public ArrayList AttributeValuesMultiString(string attributeName,
     string objectDn, ArrayList valuesCollection, bool recursive)
{
    DirectoryEntry ent = new DirectoryEntry(objectDn);
    PropertyValueCollection ValueCollection = ent.Properties[attributeName];
    IEnumerator en = ValueCollection.GetEnumerator();

    while (en.MoveNext())
    {
        if (en.Current != null)
        {
            if (!valuesCollection.Contains(en.Current.ToString()))
            {
                valuesCollection.Add(en.Current.ToString());
                if (recursive)
                {
                    AttributeValuesMultiString(attributeName, "LDAP://" +
                    en.Current.ToString(), valuesCollection, true);
                }
            }
        }
    }
    ent.Close();
    ent.Dispose();
    return valuesCollection;
}
```

Enumerate Single String Attribute Values of an Object

C#

```
public string AttributeValuesSingleString
    (string attributeName, string objectDn)
{
    string strValue;
    DirectoryEntry ent = new DirectoryEntry(objectDn);
    strValue = ent.Properties[attributeName].Value.ToString();
    ent.Close();
    ent.Dispose();
    return strValue;
}
```

Enumerate an Object's Properties: The Ones with Values

C#

```
public static ArrayList GetUsedAttributes(string objectDn)
{
    DirectoryEntry objRootDSE = new DirectoryEntry("LDAP://" + objectDn);
    ArrayList props = new ArrayList();

    foreach (string strAttrName in objRootDSE.Properties.PropertyNames)
    {
        props.Add(strAttrName);
    }
    return props;
}
```

Get an Object DistinguishedName: ADO.NET search (ADVANCED)

This method is the glue that ties all the methods together since most all the methods require the consumer to provide a `distinguishedName`. Wherever you put this code, you must ensure that you add these enumerations as well. This allows the consumers to specify the type of object to search for and whether they want the `distinguishedName` returned or the `objectGUID`.

C#

```
public enum objectClass
{
    user, group, computer
}
public enum returnType
{
 distinguishedName, ObjectGUID
}
```

A call to this class might look like:

```
myObjectReference.GetObjectDistinguishedName(objectClass.user, returnType.ObjectGUID,
"john.q.public", "contoso.com")
```

C#

```
public string GetObjectDistinguishedName(objectClass objectCls,
    returnType returnValue, string objectName, string LdapDomain)
{
    string distinguishedName = string.Empty;
    string connectionPrefix = "LDAP://" + LdapDomain;
    DirectoryEntry entry = new DirectoryEntry(connectionPrefix);
    DirectorySearcher mySearcher = new DirectorySearcher(entry);

    switch (objectCls)
    {
```

```
        case objectClass.user:
            mySearcher.Filter = "(&(objectClass=user)
        (|(cn=" + objectName + ")(sAMAccountName=" + objectName + ")))";
            break;
        case objectClass.group:
            mySearcher.Filter = "(&(objectClass=group)
        (|(cn=" + objectName + ")(dn=" + objectName + ")))";
            break;
        case objectClass.computer:
            mySearcher.Filter = "(&(objectClass=computer)
            (|(cn=" + objectName + ")(dn=" + objectName + ")))";
            break;
    }
    SearchResult result = mySearcher.FindOne();

    if (result == null)
    {
        throw new NullReferenceException
        ("unable to locate the distinguishedName for the object " +
        objectName + " in the " + LdapDomain + " domain");
    }
    DirectoryEntry directoryObject = result.GetDirectoryEntry();
    if (returnValue.Equals(returnType.distinguishedName))
    {
        distinguishedName = "LDAP://" + directoryObject.Properties
            ["distinguishedName"].Value;
    }
    if (returnValue.Equals(returnType.ObjectGUID))
    {
        distinguishedName = directoryObject.Guid.ToString();
    }
    entry.Close();
    entry.Dispose();
    mySearcher.Dispose();
    return distinguishedName;
}
```

Convert distinguishedName to ObjectGUID

C#

```
public string ConvertDNtoGUID(string objectDN)
{
    //Removed logic to check existence first

    DirectoryEntry directoryObject = new DirectoryEntry(objectDN);
    return directoryObject.Guid.ToString();
}
```

Convert an ObjectGUID to OctectString: The Native ObjectGUID

C#

```
public static string ConvertGuidToOctectString(string objectGuid)
{
    System.Guid guid = new Guid(objectGuid);
    byte[] byteGuid = guid.ToByteArray();
    string queryGuid = "";
    foreach (byte b in byteGuid)
    {
        queryGuid += @"\" + b.ToString("x2");
    }
    return queryGuid;
}
```

Search by ObjectGUID or convert ObjectGUID to distinguishedName

C#

```
public static string ConvertGuidToDn(string GUID)
{
      DirectoryEntry ent = new DirectoryEntry();
      String ADGuid = ent.NativeGuid;
      DirectoryEntry x = new DirectoryEntry("LDAP://{GUID=" + ADGuid + ">");
          //change the { to <>

      return x.Path.Remove(0,7); //remove the LDAP prefix from the path

}
```

Publish Network Shares in Active Directory

C#

```
 //Example

 private void init()
{
    CreateShareEntry("OU=HOME,dc=baileysoft,dc=com",
        "Music", @"\\192.168.2.1\Music", "mp3 Server Share");
    Console.ReadLine();
}

//Actual Method

public void CreateShareEntry(string ldapPath,
    string shareName, string shareUncPath, string shareDescription)
{
    string oGUID = string.Empty;
    string connectionPrefix = "LDAP://" + ldapPath;
    DirectoryEntry directoryObject = new DirectoryEntry(connectionPrefix);
    DirectoryEntry networkShare = directoryObject.Children.Add("CN=" +
        shareName, "volume");
    networkShare.Properties["uNCName"].Value = shareUncPath;
    networkShare.Properties["Description"].Value = shareDescription;
    networkShare.CommitChanges();
```

```
    directoryObject.Close();
    networkShare.Close();
}
```

Create a New Security Group

Note: by default if no `GroupType` property is set, the group is created as a domain security group.

C#

```
public void Create(string ouPath, string name)
{
    if (!DirectoryEntry.Exists("LDAP://CN=" + name + "," + ouPath))
    {
        try
        {
            DirectoryEntry entry = new DirectoryEntry("LDAP://" + ouPath);
            DirectoryEntry group = entry.Children.Add("CN=" + name, "group");
            group.Properties["sAmAccountName"].Value = name;
            group.CommitChanges();
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message.ToString());
        }
    }
    else { Console.WriteLine(path + " already exists"); }
}
```

Delete a group

C#

```
public void Delete(string ouPath, string groupPath)
{
    if (DirectoryEntry.Exists("LDAP://" + groupPath))
    {
        try
        {
            DirectoryEntry entry = new DirectoryEntry("LDAP://" + ouPath);
            DirectoryEntry group = new DirectoryEntry("LDAP://" + groupPath);
            entry.Children.Remove(group);
            group.CommitChanges();
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message.ToString());
        }
    }
    else
    {
        Console.WriteLine(path + " doesn't exist");
```

```
    }
}
```

Active Directory Users Tasks

C#

```
//These methods require these imports

//You must add a references in your project as well

using System.DirectoryServices;
```

Authenticate a User Against the Directory

Per John Storer, thanks for sharing.

C#

```
private bool Authenticate(string userName,
    string password, string domain)
{
    bool authentic = false;
    try
    {
        DirectoryEntry entry = new DirectoryEntry("LDAP://" + domain,
            userName, password);
        object nativeObject = entry.NativeObject;
        authentic = true;
    }
    catch (DirectoryServicesCOMException) { }
    return authentic;
}
```

Add User to Group

C#

```
public void AddToGroup(string userDn, string groupDn)
{
    try
    {
        DirectoryEntry dirEntry = new DirectoryEntry("LDAP://" + groupDn);
        dirEntry.Properties["member"].Add(userDn);
        dirEntry.CommitChanges();
        dirEntry.Close();
    }
    catch (System.DirectoryServices.DirectoryServicesCOMException E)
    {
        //doSomething with E.Message.ToString();
```

```
    }
}
```

Remove User from Group

C#

```
public void RemoveUserFromGroup(string userDn, string groupDn)
{
    try
    {
        DirectoryEntry dirEntry = new DirectoryEntry("LDAP://" + groupDn);
        dirEntry.Properties["member"].Remove(userDn);
        dirEntry.CommitChanges();
        dirEntry.Close();
    }
    catch (System.DirectoryServices.DirectoryServicesCOMException E)
    {
        //doSomething with E.Message.ToString();

    }
}
```

Get User Group Memberships of the Logged in User from ASP.NET

C#

```
public ArrayList Groups()
{
    ArrayList groups = new ArrayList();
    foreach (System.Security.Principal.IdentityReference group in
        System.Web.HttpContext.Current.Request.LogonUserIdentity.Groups)
    {
        groups.Add(group.Translate(typeof
            (System.Security.Principal.NTAccount)).ToString());
    }
    return groups;
}
```

Get User Group Memberships

This method requires that you have the `AttributeValuesMultiString` method earlier in the article included in your class.

C#

```
public ArrayList Groups(string userDn, bool recursive)
{
    ArrayList groupMemberships = new ArrayList();
    return AttributeValuesMultiString("memberOf", userDn,
        groupMemberships, recursive);
}
```

Create User Account

C#

```
public string CreateUserAccount(string ldapPath, string userName,
    string userPassword)
{
    try
    {
        string oGUID = string.Empty;
        string connectionPrefix = "LDAP://" + ldapPath;
        DirectoryEntry dirEntry = new DirectoryEntry(connectionPrefix);
        DirectoryEntry newUser = dirEntry.Children.Add
            ("CN=" + userName, "user");
        newUser.Properties["samAccountName"].Value = userName;
        newUser.CommitChanges();
        oGUID = newUser.Guid.ToString();

        newUser.Invoke("SetPassword", new object[] { userPassword });
        newUser.CommitChanges();
        dirEntry.Close();
        newUser.Close();
    }
    catch (System.DirectoryServices.DirectoryServicesCOMException E)
    {
        //DoSomethingwith --> E.Message.ToString();

    }
    return oGUID;
}
```

Dealing with User Passwords

There are some specifics to understand when dealing with user passwords and boundaries around passwords such as forcing a user to change their password on the next logon, denying the user the right to change their own passwords, setting passwords to never expire, to when to expire, and these tasks can be accomplished using `UserAccountControl` flags that are demonstrated in the proceeding sections. Please refer to this great MSDN article: Managing User Passwords for examples and documentation regarding these features. (thanks to Daniel Ocean for identifying this resource)

C#

```
//Add this to the create account method

int val = (int)newUser.Properties["userAccountControl"].Value;
     //newUser is DirectoryEntry object

newUser.Properties["userAccountControl"].Value = val | 0x80000;
    //ADS_UF_TRUSTED_FOR_DELEGATION
```

All UserAccountControl flags

```
CONST   HEX
-------------------------------
        SCRIPT 0x0001
        ACCOUNTDISABLE 0x0002
        HOMEDIR_REQUIRED 0x0008
        LOCKOUT 0x0010
        PASSWD_NOTREQD 0x0020
        PASSWD_CANT_CHANGE 0x0040
        ENCRYPTED_TEXT_PWD_ALLOWED 0x0080
        TEMP_DUPLICATE_ACCOUNT 0x0100
        NORMAL_ACCOUNT 0x0200
        INTERDOMAIN_TRUST_ACCOUNT 0x0800
        WORKSTATION_TRUST_ACCOUNT 0x1000
        SERVER_TRUST_ACCOUNT 0x2000
        DONT_EXPIRE_PASSWORD 0x10000
        MNS_LOGON_ACCOUNT 0x20000
        SMARTCARD_REQUIRED 0x40000
        TRUSTED_FOR_DELEGATION 0x80000
        NOT_DELEGATED 0x100000
        USE_DES_KEY_ONLY 0x200000
        DONT_REQ_PREAUTH 0x400000
        PASSWORD_EXPIRED 0x800000
        TRUSTED_TO_AUTH_FOR_DELEGATION 0x1000000
```

Enable a User Account

C#

```
public void Enable(string userDn)
{
    try
    {
        DirectoryEntry user = new DirectoryEntry(userDn);
        int val = (int)user.Properties["userAccountControl"].Value;
        user.Properties["userAccountControl"].Value = val & ~0x2;
            //ADS_UF_NORMAL_ACCOUNT;

        user.CommitChanges();
        user.Close();
    }
    catch (System.DirectoryServices.DirectoryServicesCOMException E)
    {
        //DoSomethingWith --> E.Message.ToString();

    }
}
```

Disable a User Account

C#

```
public void Disable(string userDn)
{
    try
    {
        DirectoryEntry user = new DirectoryEntry(userDn);
        int val = (int)user.Properties["userAccountControl"].Value;
        user.Properties["userAccountControl"].Value = val | 0x2;
             //ADS_UF_ACCOUNTDISABLE;

        user.CommitChanges();
        user.Close();
    }
    catch (System.DirectoryServices.DirectoryServicesCOMException E)
    {
        //DoSomethingWith --> E.Message.ToString();

    }
}
```

Unlock a User Account

C#

```
public void Unlock(string userDn)
{
    try
    {
        DirectoryEntry uEntry = new DirectoryEntry(userDn);
        uEntry.Properties["LockOutTime"].Value = 0; //unlock account

        uEntry.CommitChanges(); //may not be needed but adding it anyways

        uEntry.Close();
    }
    catch (System.DirectoryServices.DirectoryServicesCOMException E)
    {
        //DoSomethingWith --> E.Message.ToString();

    }
}
```

Alternate Lock/Unlock Account

It's hard to find code to lock an account. Here is my code to lock or unlock an account. `dEntry` is class variable already set to a user account. Shared by dextrous1.

C#

```
/// <summary>
/// Gets or sets a value indicating if the user account is locked out
/// </summary>
public bool IsLocked
{
```

```
    get { return Convert.ToBoolean(dEntry.InvokeGet("IsAccountLocked")); }
    set { dEntry.InvokeSet("IsAccountLocked", value); }
}
```

Reset a User Password

C#

```
public void ResetPassword(string userDn, string password)
{
    DirectoryEntry uEntry = new DirectoryEntry(userDn);
    uEntry.Invoke("SetPassword", new object[] { password });
    uEntry.Properties["LockOutTime"].Value = 0; //unlock account

    uEntry.Close();
}
```

Rename an Object

C#

```
public static void Rename(string objectDn, string newName)
{
    DirectoryEntry child = new DirectoryEntry("LDAP://" + objectDn);
   child.Rename("CN=" + newName);
}
```

Conclusion

I would have liked to include a sample project but my professional code is so tightly integrated with customer proprietary code that it was not feasible at this time.

UPDATE

If you would like to see an extremely simple implementation of some of this code check out the DirectoryServicesBrowserDialog I posted some time ago. This should demonstrate to those of you who are having trouble adding the proper references or having other difficulties.

I hope this helps out all those programmers that were like me and had to get up to speed really quickly and lost countless hours studying the `System.DirectoryServices` assembly trying to dig up answers on how to do AD tasks.

If you have some additional segments of code that are small but efficient that you'd like to include send them to me and I'll add them to this document.

History

- Originally submitted - 22 March 2007
- Added `GetUsedAttributes()` method - 24 March 2007
- Added `EnumerateOU()` method - 25 March 2007
- Added `CreateShareEntry()` - 27 March 2007
- Added `CommitChanged()` call in `UnlockUserAccount()`
- Cleaned up article, added a few new methods - 03 Apr 2007
- Added note on `DirectoryEntry` - 04 Apr 2007
- Added note on working with local accounts - 12 Apr 2007
- Added code for creating/deleting groups - 20 Apr 2007
- Added John Storer's code for authenticating users against the directory
- Added `UserAccountControl` flags section - 06 Jun 2007
- Added Managing IIS with `DirectoryEntry()` section - 12 Jun 2007
- Created table of contents - 09 Jul 2007
- Added Dealing with User Passwords Section - 21 May 2008
- Added Alternate Lock/Unlock Account by dextrous1 - 21 May 2008

Setting UserAccountControl flags

License

Written By

thund3rstruck

Software Developer

United States

I'm a professional .NET software developer and proud military veteran. I've been in the software business for 20+ years now and if there's one lesson I have learned over the years, its that in this industry you have to be prepared to be humbled from time to time and never stop learning!

Comments and Discussions

439 messages have been posted for this article Visit **https://www.codeproject.com/Articles/18102/Howto-Almost-Everything-In-Active-Directory-**

via-C to post and view comments on this article, or click **here** to get a print view with messages.