



realtimepublishers.comtm

*The Administrator
Shortcut Guidetm To*



**VBScripting for
Windows**

Don Jones

Chapter 2: Working with ADSI24

ADSI Without a Directory24

ADSI Providers27

 The WinNT Provider27

 The LDAP Provider28

An ADSI Shortcut.....30

Querying Global Catalog Servers32

Useful ADSI Scripts33

 User Account Scripts33

 Group Scripts35

 Computer Account Scripts36

 Computer Management Scripts.....37

Scripting Batch Operations39

Summary41

Copyright Statement

© 2004 Realtimerepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimerepublishers.com, Inc. (the "Materials") and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimerepublishers.com, Inc or its web site sponsors. In no event shall Realtimerepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimerepublishers.com and the Realtimerepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimerepublishers.com, please contact us via e-mail at info@realtimerepublishers.com.

Chapter 2: Working with ADSI

With all the hype about WMI, ADSI has been getting short shrift, which is a shame because ADSI is such a useful tool. A possible reason that ADSI is overlooked by most administrators is that it is poorly named. Many administrators assume that a tool named Active Directory Services Interface deals solely with AD. The truth is that ADSI *can* work with AD, but offers many additional capabilities.

ADSI Without a Directory

Some of ADSI's most useful features have nothing to do with *any* directory, let alone AD. Listing 2.1 shows a favorite sample scripts; this script should work without modification in any environment. It contacts any Windows file server—domain member or not—and tells you which users currently have a particular shared file open.

```

` first, get the server name we want to work with
sServerName = InputBox ("Server name to check")

` get the local path of the file to check
sFilename= InputBox ("Full path and filename of the " & _
"file on the " & _
"server (use the local path as if you were " & _
"at the server console)")

` bind to the server's file service
Set oFileService = GetObject("WinNT://" & sServerName & _
"/lanmanserver,fileservice")

` scan through the open resources until we
` locate the file we want
bFoundNone = True

` use a FOR...EACH loop to walk through the
` open resources
For Each oResource In oFileService.Resources

` does this resource match the one we're looking for?
On Error Resume Next
If oResource.Path = sFilename Then
If Err <> 0 Then
WScript.Echo "Couldn't locate the resource, or " & _
"permission denied."
End If
` we found the file - show who's got it
bFoundNone = False
WScript.Echo oResource.Path & " is opened by " & oResource.User
End If
Next

` if we didn't find the file open, display a msg
If bFoundNone = True Then
WScript.Echo "Didn't find that file opened by anyone."
End If

```

Listing 2.1: Sample ADSI script that enables you to determine which users have a shared file open.

This script provides a great way to learn how ADSI works outside of directories. The script starts off by asking you for a server name to check. This name should be entered without backslashes. Next, you must type the path and filename of the open file you're curious about. This path needs to be the local path on the server. For example, suppose you have a file server that contains a folder named D:\Shares\Sales. This folder is shared as SalesData, and it contains an Excel spreadsheet named Forecasts.xls. Users access this file at \\Server\SalesData\Forecasts.xls. However, you would provide this script with D:\Shares\Sales\Forecasts.xls, because that's the local path as far as the server is concerned.

Next, the script begins to use ADSI and uses the Set statement. In the previous chapter, I explained that the Set statement is used to assign an object reference to a variable. That's exactly what's going on here. However, in the previous chapter, I also told you that the CreateObject() function was used to load DLLs into memory and obtain that reference—this script is using GetObject() instead.

In the case of ADSI, you don't *want* to load a DLL into memory. You want to reach out and connect to some service that is already running on a remote machine. Thus, rather than *creating* an object, you just want to *access* an existing, remote object. Hence, GetObject(). The argument passed to GetObject() tells VBScript what to go get. In this case, the WinNT:// name tells VBScript that you're using ADSI's Windows NT directory provider. The script also passes the server name you entered, then specifies a connection to the lanmanserver object on that server. Specifically, it wants a fileservice named lanmanserver.

So, we've established that ADSI isn't just for AD. ADSI is actually a generic directory access technology. Even NT boxes expose many of their services in a directory-like format that is accessible through ADSI's WinNT provider. Let's take a closer look at the ADSI call in Listing 2.1:

```
` bind to the server's file service
Set oFileService = GetObject("WinNT://" & sServerName & _
    "/lanmanserver,fileservice")
```


In this call:

- The script uses the Set keyword because it is assigning an object reference to a variable.
- The script specifies the variable—in this case, oFileService.
- The script specifies the GetObject() function, which has nothing specifically to do with ADSI; it is simply a generic function that connects to the object you specify. Generally, this function connects to objects that are actually services running on the local machine or a remote machine.
- The script specifies the ADSI provider—in this example, WinNT://.



The provider names are case-sensitive; thus, in this case, winnt:// will *not* work.

- Next is the server name, which can be the local box or a remote one, it doesn't matter. In this example, the script uses a variable to store the server name, and appends it into the ADSI call by using VBScript's string-concatenation operator, the ampersand (&).
- Next, the script names an object or resource on the computer, which is the object or resource to which ADSI will actually connect.

 Later in this chapter, we'll explore user and group names as the object to connect with; in the example that Listing 2.1 shows, the object or resource is lanmanserver, which happens to be the internal name of Windows' Server service.

- Optionally, you can specify an object class. In this example, the script specifies fileshare. If you don't specify an object class, ADSI will latch onto the first object or resource it finds that matches the name specified. For example, if I had a user named lanmanserver, then ADSI might grab that user instead of the Server service. However, by specifying the object class, I ensure that ADSI will grab the resource in which I'm interested.

After the first 11 lines of the script, it gets less complicated. On line 15, the script assigns the value True to a variable named bFoundNone, which is a sort of reminder that I haven't yet found any users who were using the specified file.

Keep in mind that, at this point, oFileService essentially represents the Server service on whatever server the script specifies. The Server service exposes a list of every resource it is currently managing. Thus, on line 19, I walk through each one of those resources in a For Each...Next loop. Each time through the loop, oResource will represent the current resource.

On line 22, the script performs the actual comparison: Does the current resource's Path property equal the filename that you typed into the script? If so, the script sets bFoundNone to the value False, because it has clearly found someone who is using the file. Next, the script outputs the resource's path and the user name that has the file open. *The script doesn't stop the script at this point.* The reason is that than one user can have a file open, and I want to find all of them. Thus, the script has the For Each...Next loop continue through the whole *collection* of resources.

When the loop is done, the script displays a message if it hasn't found any users who have the file open. If the script didn't do so and nobody had the specified file open, the script would end with no visual indication, leaving you to wonder whether the script was still running.

How This Script Was Designed

I have been working with Windows since the NT 3.1 days. Until Win2K, administrators used a management application called Server Manager. With it, you could double-click a server and get a list of every resource that the server was currently managing—namely, open shared files. To discover which user had a file open, you simply opened this list—which on a busy file server would have thousands of entries—and start scrolling through it, looking for the filename in which you were interested. Of course, by the time you got to the end of the list, it was out of date and you would have to close the list, reopen it, and start over. However, it never took more than an hour or so to go through the list.

The script that Listing 2.1 shows performs the same task as the Server Manager utility—it simply does so faster and with an infinitely better attention span to its assigned task. In designing this script, I simply wanted to duplicate—and speed—the task I already knew how to do manually. With a little research, I discovered that ADSI could perform this task for me.

ADSI Providers

Now that we've explored a little about how ADSI works, it's probably time for a more formal introduction. ADSI is Microsoft's technology for generic directory access; the fact that ADSI has Active Directory in the name does *not* mean that it works only with AD. ADSI is preinstalled on Win2K and later, and its core components are also installed with the Directory Services client for NT. ADSI ships with three key providers:

- WinNT—Provides access to NT domains, local computer resources (including Win2K and later boxes), and local security accounts (for Win2K and later).
- Lightweight Directory Access Protocol (LDAP)—Enables you to connect to any LDAP-enabled directory, such as Lotus Notes/Domino, Novell Directory Services (NDS), Microsoft Exchange Server 5.0 or 5.5, and Microsoft AD.

 There is also a specific NDS provider, but LDAP is just as useful with the latest versions of NDS.

- NWCOMPAT—Provides connectivity to Novell NetWare 2.x and 3.x binderies (should you happen to work in the Smithsonian where they surely have some of these relics on display).

All of the providers work slightly differently. And let me emphasize once again that these are *case-sensitive*. In other words, ldap is *not* the same thing as LDAP, and ADSI will give vague errors if you use the incorrect provider name.

The WinNT Provider

You've already seen the basic syntax for the WinNT provider:

```
WinNT://host/object,class
```

Host in this case can be a server, workstation, or domain name. If you provide a domain name, your client will follow all the usual rules for locating a domain controller (for example, querying WINS, querying DNS, and so on). Specifying a domain name is useful if you want to mess around with domain usernames and groups; you can also specify the name of a domain controller, of course, rather than the domain name. It's your choice. If you perform an operation that requires an object to be changed (such as changing a user's password), then ADSI will automatically reconnect to the Primary Domain Controller (PDC), the only domain controller in an NT domain that can make changes to the domain (remember that Backup Domain Controllers—BDCs—are read-only).

Pay close attention: *You can connect to AD by using the WinNT provider*. The reason is that AD *emulates* an NT domain. There is a PDC Emulator that runs on one domain controller in every AD domain, and every AD domain controller can provide services to downlevel, NT-based clients.

The LDAP Provider

NT domains are a flat namespace, which is a fancy way of saying that there are no organizational units (OUs), containers, sites, and so forth. Thus, if you're connecting to AD via the WinNT provider, you can't modify OUs, containers, sites, and the things that NT domains don't have. To use the full power of AD through ADSI, you must use the slightly more complex LDAP provider, which works like this:

```
LDAP://fqdn
```

Not very exciting, is it? Here's an actual example:

```
LDAP://cn=donj,ou=research,dc=scriptinganswers,dc=com
```

You have to use LDAP-style naming to provide the *fully-qualified domain name* (FQDN) of the object to which you want to connect. There is no need to specify object classes because you're being very specific: the example shows a user named donj, in an OU named research, in a domain named scriptinganswers.com. The order of these components is important. You must start with the object name, then the OU that contains the object, any parent OUs, then the domain name components from left to right (for example, scriptinganswers first, then com second, for scriptinganswers.com).

When you plug this query into `GetObject()`, the output depends on the query. If you query a user, the output will be a user object. If you query a computer, you'll get a computer object back. However, you can't query non-directory items, such as lanmanserver; although the WinNT provider has access to a number of non-directory items, LDAP can only access what is in the directory. What you can do with the object that `GetObject()` returns depends on what type of object it is. For example, with a user object, you can:

```
Set oUser = GetObject _
    ("LDAP://cn=donj,ou=research,dc=scriptinganswers,dc=com")
oUser.ChangePassword "password", "BetterPassword!"
```

In this example, the User object, as you can see, sports a `ChangePassword` method that can be used to change the password. You must know the old password in order to change it to a new password.

If you query a Group object, you can change the group's type:

```
Const ADS_GROUP_TYPE_GLOBAL_GROUP = &h2
Const ADS_GROUP_TYPE_LOCAL_GROUP = &h4
Const ADS_GROUP_TYPE_UNIVERSAL_GROUP = &h8
Const ADS_GROUP_TYPE_SECURITY_ENABLED = &h80000000

Set oGroup = GetObject _
    ("LDAP://cn=Writers,dc=scriptinganswers,dc=com")
oGroup.Put "groupType", _
    ADS_GROUP_TYPE_GLOBAL_GROUP + ADS_GROUP_TYPE_SECURITY_ENABLED

oGroup.SetInfo
```

In this example, the script first defines four constants. Constants, as you may remember, are a way of assigning a friendly name to a difficult-to-remember value. They also make your script a bit easier to read; in this case, the names are easier to figure out than the hexadecimal values that the names represent.

Next, the script uses an LDAP call to ADSI to retrieve a group named Writers. The script then uses the Put method of the Group object. Most objects returned by GetObject() from an LDAP query will support the Put and Get methods. These methods allow you to write and read, respectively, any property of the object. In the previous example, the script is writing the groupType property, which, not surprisingly, tells AD what type of group this is.

After specifying the property name, the script needs to specify the value to be put into the property. All you need to do is specify the correct constant name, and if you want the group to be a security group (as opposed to a distribution group), add that constant into the mix. In this example, the group is a global security group.

Whenever you use Put to make changes to an object's properties (or *attributes*), you must finish off with the SetInfo method to save your changes. The reason is that ADSI caches your changes as you make them, but doesn't actually send them off to the domain controller until you execute SetInfo. This technique helps make ADSI a bit more efficient.

An ADSI Shortcut

The Microsoft Scripting Guys—Microsoft employees who promote scripting in their spare time, and who are also full-time members of the Windows product team—have created several labor-saving devices for scripters. One of their nifty tools is the ADSI Scriptomatic, which you can download for free from

<http://www.microsoft.com/technet/community/scriptcenter/tools/admatic.msp>. It's an HTML Application (HTA), which means you'll need to have Internet Explorer (IE) installed to run it. Figure 2.1 shows the ADSI Scriptomatic.

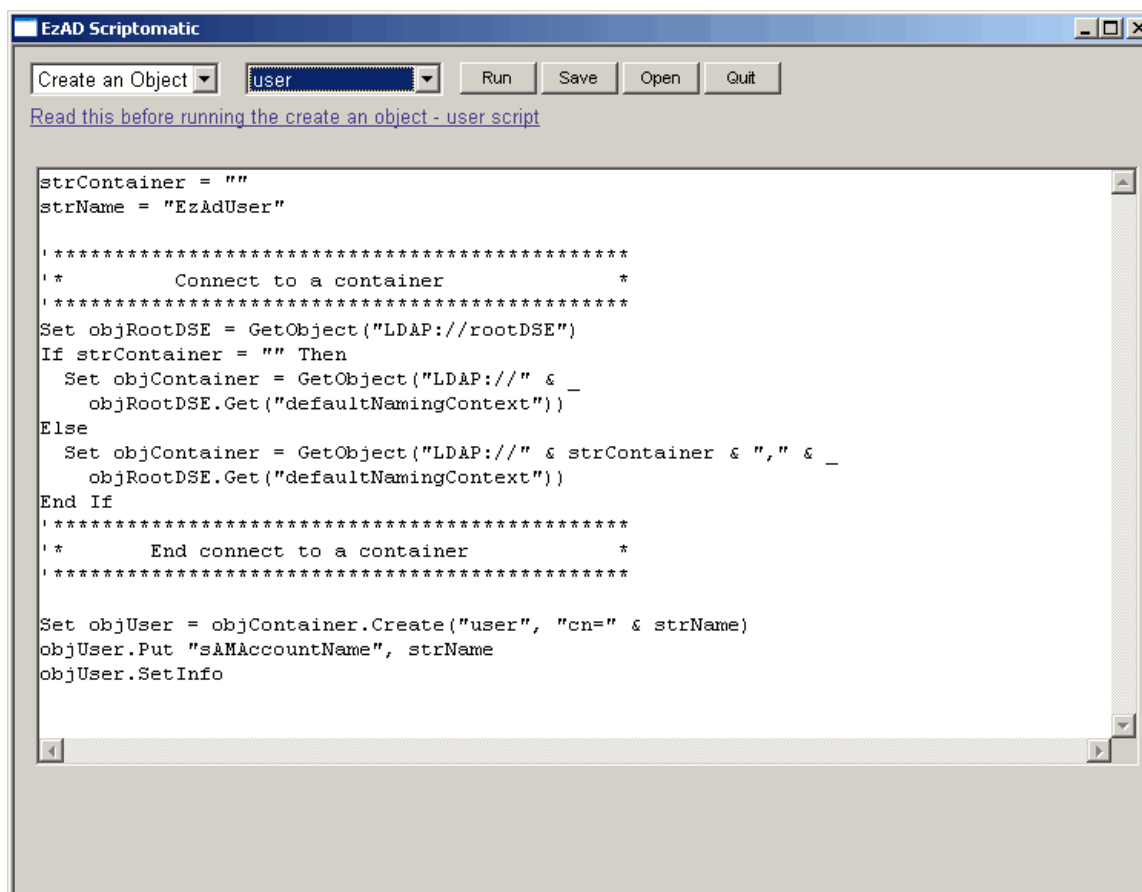



Figure 2.1: The ADSI Scriptomatic generates ADSI scripts for you.

Basically, you just tell this tool what you want to do—Create an Object, in this case—and what object class you want to use. I selected the User class, so the tool generated a script to create a user. Let me briefly walk through this script, because it's the same basic template used for all of the tool's scripts. Understanding it will help you better understand how ADSI works.

The first two lines simply set up two string variables:

```
strContainer = ""
strName = "EzAdUser"
```


These contain the name of the container in which the new object should be created, and the name that the new object should be given. You would obviously change these to the appropriate names for your environment.

 Notice that the script uses a variable naming convention; per best practice, the tool uses a three-letter prefix as specified in Hungarian notation, which is a Microsoft-favored notation for VBScript.

Next, as the comment says, the script connects to the container. First, it connects to the root of AD, for whichever domain the computer running the script is a member:

```
\*****
\*           Connect to a container           *
\*****

Set objRootDSE = GetObject("LDAP://rootDSE")
```

 rootDSE is a special LDAP object that exists on all LDAP servers. Any LDAP server (version 3 or later) will respond correctly to a query for the rootDSE.

If a container name has not been specified, the script connects to the default naming context of the domain's root. Otherwise, the script connects to the default naming context of the specified container. Either way, what you get back is a container of some kind—the default Users container or a specified OU:

```
If strContainer = "" Then
    Set objContainer = GetObject("LDAP://" & _
        objRootDSE.Get("defaultNamingContext"))
Else
    Set objContainer = GetObject("LDAP://" & strContainer & "," & _
        objRootDSE.Get("defaultNamingContext"))
End If


\*****
\*           End connect to a container           *
\*****
```

Next, the script uses the container's `Create` method to create an object. The script specifies the class of the object as `user` and specifies the new user's common name, which is `cn=` plus whatever is in the variable `strName`. The `Create` method returns a reference to the newly created object, which is stored in the variable `objUser`.

```
Set objUser = objContainer.Create("user", "cn=" & strName)
```

Next, the script uses the new object's `Put` method to set the `sAMAccountName` attribute to the new object's username. `SetInfo` is called to save everything, and the script is finished.

```
objUser.Put "sAMAccountName", strName
objUser.SetInfo
```


 `sAMAccountName` might look like a case of the Shift key gone wrong, but it's actually correct. SAM stands for Security Accounts Manager; the attribute *should* be named "SAMAccountName," but AD's schema naming standards specify a lowercase letter for the first letter in the attribute name, so `sAMAccountName` it is.

Querying Global Catalog Servers

In an AD domain, one or more servers fill the role of Global Catalog (GC) server, a server that holds information about every object in the entire forest. You can query GCs directly by using ADSI. The following example uses the special GC: provider to get a list of GC servers' domains:

```
Set oList = GetObject("GC:")
For Each oServer in oList
    WScript.Echo oServer.ADsPath
Next
```

Of course, once you have a GC's name (or `ADsPath`), you can query the server directly to retrieve information.

 Microsoft provides a sample script at <http://support.microsoft.com/?kbid=252490> that will tell you whether a particular user principle name (UPN) exists in a forest so that you can avoid making duplicate user objects.

Useful ADSI Scripts

At this point, you should have some idea of how ADSI works and how you can use it within your scripts. The best way to learn more details is to actually explore useful ADSI scripts, so I'll give you a bunch of short little scriptlets and explain how they work. Most of these are scriptlets that you would incorporate into your own, larger scripts; they're not necessarily intended to be run entirely on their own. Pay special attention to where I'm using the WinNT and LDAP providers; you'll get a better feel for what capabilities each can provide to your scripts.

User Account Scripts

The ability to batch create users is a great reason to write a script. Suppose you have a text file (or an Excel spreadsheet) that lists the users you need to create. From Excel, save the file as a CSV and use the script that Listing 2.2 shows.

```

` PART 1: Open up the text file
Dim oFSO, oTS
Set oFSO = CreateObject("Scripting.FileSystemObject")
Set oTS = oFSO.OpenTextFile("c:\users.csv")

`let's agree that the CSV file's first line will
`be the column names, and that they will be:
` Username, Fullname, Description, HomeDir

` PART 2: Get a reference to the
` Windows NT or AD domain using ADSI
Dim oDomain
Set oDomain = GetObject("WinNT://NT4PDC")

` PART 3: Open an output text file
` to store users' initial passwords
Dim oTSOut
Set oTSOut = oFSO.CreateTextFile("c:\passwords.csv",True)

` create the necessary variables
Dim sUserID, sFullName, sDescription
Dim sPassword, sHomeDir

`skip the first line of the file
Dim sLine, sData
sLine = oTS.ReadLine

` now go through the file one
` line at a time
Do Until oTS.AtEndOfStream

`read the line and split it
`on the commas
sLine = oTS.ReadLine
sData = Split(sLine,",")

` get the user information from this row

```

```

sUserID = sData(0)
sFullName = sData(1)
sDescription = sData(2)
sHomeDir = sData(3)

` make up a new password
sPassword = Left(sUserID,2) & _
    DatePart("n",Time) & _
    DatePart("y",Date) & _
    DatePart("s",Time)

` create the user account
Set oUserAcct = oDomain.Create("user",sUserID)
oUser.Put "sAMAccountName", sUserID

` set account properties
oUserAcct.SetPassword sPassword
oUserAcct.FullName = sFullName
oUserAcct.Description = sDescription
oUserAcct.HomeDirectory = sHomeDir

` save the account
oUserAcct.SetInfo


` write password to file
oTSOut.Write sUserID & "," & sPassword & VbCrLf

Loop

` PART 6: Final clean up, close down
oRS.Close
oTS.Close
WScript.Echo "Passwords have been written to c:\passwords.csv."

```

Listing 2.2: Script to batch create users.

 This example script employs fancy stuff with text files that I haven't yet introduced; we'll explore them a bit later in this chapter.

The script that Listing 2.2 shows assumes that your column names are Username, Fullname, Description, and HomeDir, and that the first row of your spreadsheet (or CSV file) uses those column names. Because it utilizes the WinNT provider, this script will work with any Windows domain; in an AD domain, users will be created in the default Users container, and you can move them from there. The script makes up a non-random password for each user, which it writes to a text file for your reference.

Group Scripts

One common task that you might want to perform for a group is to determine a group's membership, especially with security-sensitive groups such as the local Administrators group. In this case, the WinNT provider can be helpful, even in AD domains, because the WinNT provider can work with local groups on member and standalone computers. The following script provides a list of every local group on a designated computer and every member of each group:

```
Dim sComputer, cGroups, oGroup, oUser
sComputer = "ClientA"
Set cGroups = GetObject("WinNT://" & sComputer & "")
cGroups.Filter = Array("group")
For Each oGroup in cGroups
    WScript.Echo oGroup.Name & " members:"
    For Each oUser in oGroup.Members
        WScript.Echo oUser.Name
    Next
Next
```

This script connects to the remote computer and retrieves every object that the computer's WinNT provider can handle. It then adds a one-element array containing the string "group" to the resulting collection's Filter property. Doing so has the effect of filtering the collection to only include (or at least *seem* to include) objects of the class "group." Then the script uses two For Each...Next loops. The outer loop enumerates the groups on the computer, and the inner loop enumerates the members of each group.

If you want to add a user to a group, you can use a script similar to the following example. This script uses ADSI to add a user to an AD group:

```
Const ADS_PROPERTY_APPEND = 3

Set objGroup = GetObject _
    ("LDAP://cn=Special,cn=Users,dc=scriptinganswers,dc=com")

objGroup.PutEx ADS_PROPERTY_APPEND, "member", _
    Array("cn=donj,ou=Executives,dc=scriptinganswers,dc=com")

objGroup.SetInfo
```

This script starts by getting a reference to the group that is to be modified. Keep in mind that the members of a group are part of the *group* object, not the user. In other words, user objects don't contain a list of the groups to which the user belongs. Group objects contain lists of users (and other groups), not the other way around. Thus, if you want to modify a group's membership, you must access the group object.

Next, the script modifies the group's "member" property. Rather than completely overwrite this property, the script simply seeks to append a value to the property. The special PutEx method allows you to pass the property a value, which is stored in a constant, specifying that the script is only appending data to the property, not completely rewriting the property.

The script then provides the property in question—"member"—and the value to be appended. PutEx takes arrays, so the script uses the VBScript Array() function. This function accepts a list of values (just one value, in this example, but you could provide a whole comma-delimited list of values) and turns the list into an array that PutEx can deal with. The value that the script provides is the FQDN of the user to be added to the group (you can also specify a group to add to the group). Finally, the script uses SetInfo to write everything back to the domain controller and saves the change.

Computer Account Scripts

AD does amazingly cool stuff when a Win2K or later machine logs on. The NetLogon service actually feeds some basic inventory data about the machine into AD—kind of a mini-Systems Management Server (SMS). Some of this information is displayed in the computer's Properties dialog box in the Microsoft Management Console (MMC) Active Directory Users and Computers console; you can also query this information from AD by using a script similar to the example that Listing 2.3 shows.

```
strContainer = "ou=Domain Controllers"
strName = "BRAINCORENET"

On Error Resume Next

\*****
\*          Connect to an object          *
\*****
Set objRootDSE = GetObject("LDAP://rootDSE")
If strContainer = "" Then
    Set objItem = GetObject("LDAP://" & _
        objRootDSE.Get("defaultNamingContext"))
Else
    Set objItem = GetObject("LDAP://cn=" & strName & "," & _
        strContainer & "," & _
        objRootDSE.Get("defaultNamingContext"))
End If
\*****
\*          End connect to an object      *
\*****

WScript.Echo "    DNS: " & objItem.Get("dnsHostName")
WScript.Echo "    OS: " & objItem.Get("operatingSystem")
WScript.Echo "OS ver: " & objItem.Get("operatingSystemVersion")
WScript.Echo "    SP: " & _
    objItem.Get("operatingSystemServicePack")
WScript.Echo "Hotfix: " & objItem.Get("operatingSystemHotfix")
```

Listing 2.3: An example script to query basic inventory information about a machine.

As Listing 2.3 shows, I used the ADSI Scriptomatic as the starting point for my script, and filled in the OU and computer name that I was interested in. The Scriptomatic code connects to the computer object in the directory; the script then uses the Get method to retrieve a few properties. Try this with Win2K machines, which are interesting because they have service packs and other items installed. Querying a Windows 2003 server gets you the following output:

```
DNS: braincorenet.braincore.pri
OS: Windows Server 2003
OS ver: 5.2 (3790)
```

Notice that the last two properties aren't displayed because I haven't applied any service packs or anything to this computer.

Where Does He Get This Stuff?

You might be wondering where I discovered the "operating SystemServicePack" attribute and some of the other properties. Knowing what is available to you is a big part of scripting. In the previous chapter, I recommended becoming friendly with the VBScript documentation because browsing would help you determine what is possible with VBScript. I recommend the same thing for ADSI—get to know your objects.

You can do so by browsing the Internet. Plenty of sites, including Microsoft's and my own ScriptingAnswers.com, provide oodles of examples to learn from. The easiest way to get to know ADSI, however, is built right into Windows itself. Open a command line on a domain controller. Change to the \Windows\System32 folder. Run Regsvr32 schmmgmt.dll, then run Mmc. From the Console menu, select Add/Remove Snap-Ins, and add the Active Directory Schema snap-in (the business about running Regsvr32 is necessary because the Schema snap-in isn't registered and available by default; thus, you only need to do it the first time).

Next, click on the "computer" class in the left tree view. In the right pane, you see a list of every possible attribute for a computer object; you can do the same for any AD object, including users, groups, and more. Browsing through the list reveals plenty of potentially useful properties, all of which are accessible from your scripts.

Computer Management Scripts

The WinNT provider is very useful for working with local computer accounts. For example, one security task that often goes undone is a regular password change for the local Administrator account on each of your computers. The script that Listing 2.4 shows will read computer names (one name per line) from a text file named c:\computers.txt, then change the Administrator account password on each of them. This script will skip over any computers that the script can't contact at the time, writing their names to c:\skipped.txt. You can then rename c:\skipped.txt to c:\computers.txt and have the script retry those computers later.

```

`Create a FileSystemObject
Set oFS = CreateObject("Scripting.FileSystemObject")

`Open a text file of computer names
`with one computer name per line
Set oTS = oFS.OpenTextFile("c:\computers.txt")
Set oTSOut = oFS.CreateTextFile("c:\skipped.txt")

`go through the text file
Do Until oTS.AtEndOfStream

  `get the next computer name
  sComputer = oTS.ReadLine

  `retrieve that computer's local Admin
  On Error Resume Next
  Set oUser = GetObject("WinNT://" & _
    sComputer & "/Administrator,user")

  If Err = 0 Then
    `reset the password
    oUser.SetPassword "New!Ad3in"

    `save the change
    oUser.SetInfo
  Else
    oTSOut.WriteLine sComputer
  End If
  On Error Goto 0

Loop

`close the text file
oTS.Close
oTSOut.Close
MsgBox "Done!"

```

Listing 2.4: Example script to change the password for the local Administrator account on each of your computers.

You can see the `FileSystemObject` object in action; you learned about that object in the previous chapter. As you can see, the script reads through an entire text file, which is represented by the variable `oTS`. The object type for that is called a `TextStream`, because a text file is really just a big stream of text characters. One of the properties of a `TextStream` object is `AtEndOfStream`, which is set to the value `True` when the end of the file is reached. Thus, the script's `Do Until` loop will continue reading the text file until the end of the file is reached.

The `ReadLine` method of the `TextStream` is used to read a line of text and store it in the variable `sComputer`. That variable, and ADSI's `WinNT` provider, are used to connect to the specified computer's Administrator account. What comes back is a `User` object, and the script uses its `SetPassword` method to set a new password. The script then calls `SetInfo` just to be sure everything is saved, although technically with `SetPassword`, `SetInfo` isn't necessary.

Notice the On Error Resume Next object. This object tells VBScript to keep running the script even if it encounters an error, which prevents VBScript from stopping the script if it tries to connect to a computer that is currently turned off. Right after it tries to connect, the script checks the value of the special built-in Err object. If the value of this object is zero, no error occurred and the script can perform the password change. If Err is something other than zero, an error occurred, and the script writes the offending computer's name out to a second TextStream, which is c:\skipped.txt. On Error Goto 0 tells VBScript to start paying attention to errors again.

Scripting Batch Operations

A *batch operation* is any operation that needs to run over and over. For example, creating one hundred users in a script is a good example of a batch operation, and is certainly a task you would rather script than perform manually. The first example I showed you, in which you use a script to determine which users have a shared file open, is another batch operation.

On my Web site at <http://www.ScriptingAnswers.com>, I provide several scripts that perform batch ADSI operations. One of the most useful is a generic script template that can be used to perform some operation against every user and/or computer account in a domain (see Listing 2.5).

```
'connect to the root of AD
Dim rootDSE, domainObject
Set rootDSE=GetObject("LDAP://RootDSE")
domainContainer = rootDSE.Get("defaultNamingContext")
Set oDomain = GetObject("LDAP://" & domainContainer)

'start with the domain root
WorkWithObject(oDomain)

Sub WorkWithObject(oContainer)
Dim oADObject
For Each oADObject in oContainer
  Select Case oADObject.Class
    Case "user"
      'oADObject represents a USER object;
      'do something with it
      '** YOUR CODE HERE**
    Case "computer"
      'oADObject represents a COMPUTER object;
      'do something with it
      '** YOUR CODE HERE**
    Case "organizationalUnit" , "container"
      'oADObject is an OU or container...
      'go through its objects
      WorkWithObject(oADObject)
  End select
Next
End Sub
```

Listing 2.5: A generic script template that you can use to perform an operation against every user and/or computer account in a domain.

You can see where in Listing 2.5 you would insert your own code. You will use the variable `oADObject`, which represents either a computer object or a user object. The current OU is always represented by `oContainer`, and this script will iterate through each and every OU in the domain and find every computer or user within.

For example, suppose your company moved its offices, and you wanted to write a script that changed every user account's ZIP code to 98053. You could use the template script with just a couple of additional lines of code, as Listing 2.6 shows.

```

`connect to the root of AD
Dim rootDSE, domainObject
Set rootDSE=GetObject("LDAP://RootDSE")
domainContainer = rootDSE.Get("defaultNamingContext")
Set oDomain = GetObject("LDAP://" & domainContainer)

`start with the domain root
WorkWithObject(oDomain)

Sub WorkWithObject(oContainer)
Dim oADObject
For Each oADObject in oContainer
Select Case oADObject.Class
Case "user"
`oADObject represents a USER object;
`do something with it
oADObject.Put "postalCode", "98053"
oADObject.SetInfo
Case "computer"
`oADObject represents a COMPUTER object;
`do something with it
`** YOUR CODE HERE**
Case "organizationalUnit" , "container"
`oADObject is an OU or container...
`go through its objects
WorkWithObject(oADObject)
End select
Next
End Sub

```

Listing 2.6: Using the generic template as a starting point for a script to change users' zip code.

I've boldfaced the two additional lines. Notice that I didn't add any code to the section that deals with computer accounts, meaning computers won't be changed—only users.

Another example is to create a text file listing each user and the user's OU. Combining what you learned about the FileSystemObject (in the previous chapter), you might modify the template script as Listing 2.7 shows.

```

`connect to the root of AD
Dim rootDSE, domainObject
Set rootDSE=GetObject("LDAP://RootDSE")
domainContainer = rootDSE.Get("defaultNamingContext")
Set oDomain = GetObject("LDAP://" & domainContainer)

`create a file
Dim oFSO, oTS
Set oFSO = CreateObject("Scripting.FileSystemObject")
Set oTS = oFSO.CreateTextFile("c:\output.txt",True)

`start with the domain root
WorkWithObject(oDomain)

oTS.Close
MsgBox "All done, boss."

Sub WorkWithObject(oContainer)
Dim oADObject
For Each oADObject in oContainer
  Select Case oADObject.Class
    Case "user"
      `oADObject represents a USER object;
      `do something with it
      oTS.WriteLine oADObject.Get("distinguishedName")
    Case "computer"
      `oADObject represents a COMPUTER object;
      `do something with it
      `** YOUR CODE HERE**
    Case "organizationalUnit" , "container"
      `oADObject is an OU or container...
      `go through its objects
      WorkWithObject(oADObject)
  End select
Next
End Sub

```

Listing 2.7: Using the generic template as a starting point to list each user and user's OU.

This script will write each user's FQDN to a text file named c:\output.txt. Hopefully, this batch template will prove useful in helping you write your own batch ADSI scripts.

Summary

In this chapter, I've introduced you to ADSI. You learned about two key providers, WinNT and LDAP, and how to use them. In addition, we explored how to create new directory objects, connect to services on remote machines, and more. ADSI is a large part of the two-part approach to improved Windows administration through scripting. ADSI provides remote administration capabilities, batch processing capabilities, and more. The other half is WMI, which I'll cover in the next chapter.